68' MICRO JOURNAL

Australia A $7.95 | New Zealand NZ $9.85
Singapore S $14.95 | Hong Kong H $29.00
Malaysia M $14.25 | Sweden 30:-SEK

$2.95USA

OS-9 Atari Amiga Mac S-50

6800 6809 68008 68000 68010 68020 68030

The Magazine for Motorola CPU Devices *For Over a Decade!*

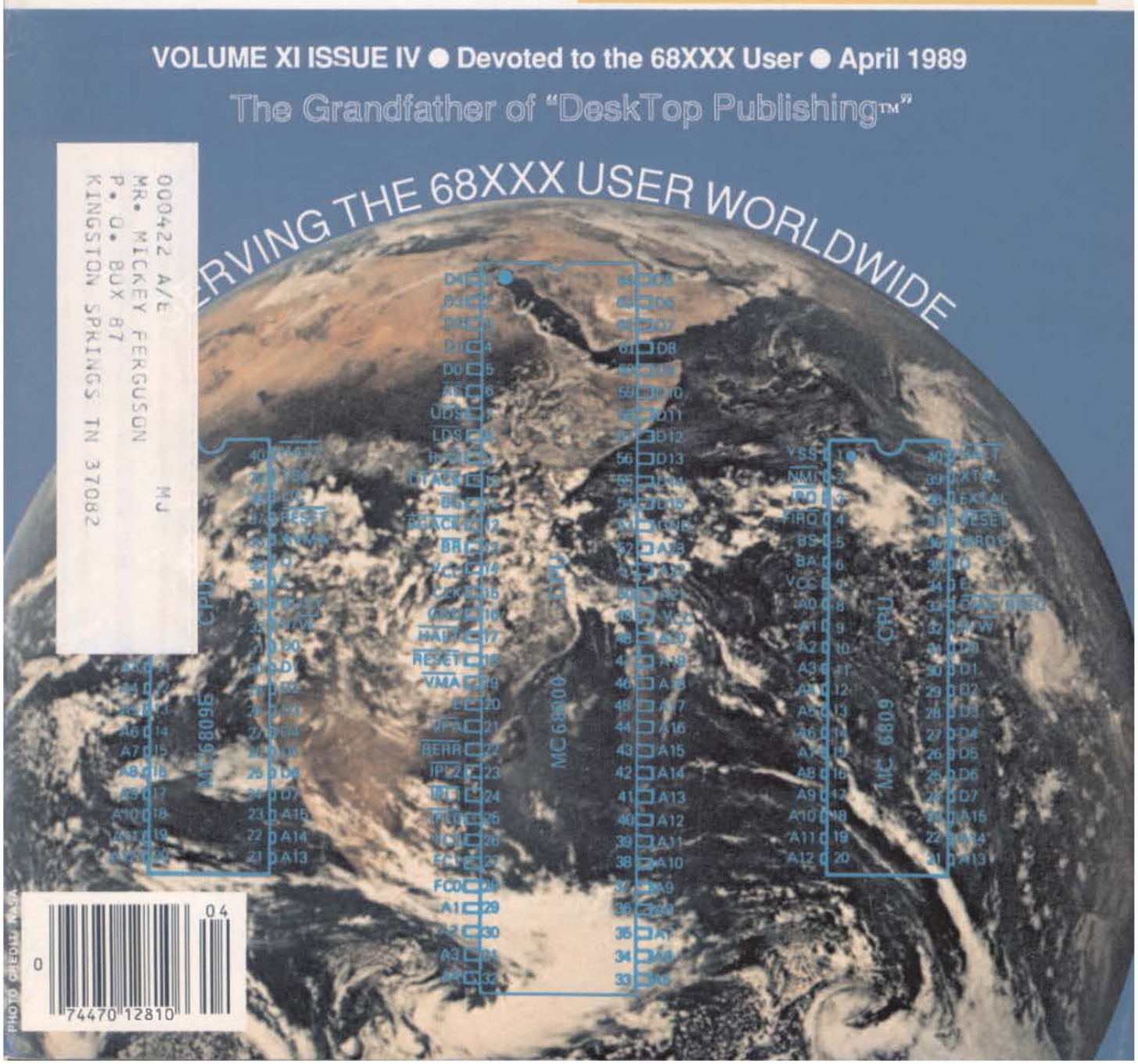
This Issue:

OS-9 SK*DOS Atari Amiga FLEX Macintosh *A User Contributor Journal*

And Lots More!

VOLUME XI ISSUE IV ● Devoted to the 68XXX User ● April 1989

The Grandfather of "DeskTop Publishing™"



0 74470 12810 04

WHO DO YOU CALL WHEN YOUR DEBUGGER WON'T DEBUG?

The problem with most real-time operating systems is simple, they're not an integrated solution. You end up dealing with a multitude of suppliers for languages, compilers, debuggers and other important development tools. And when something does go wrong, it can be a frustrating experience trying to straighten out the mess.

Why Not Try the Microware One-Stop Total Solution?

Microware's OS-9 Real-Time Operating System is a total integrated software system, not just a kernel. We offer an extensive set of development tools, languages, I/O and Kernel options. ***And this total integrated solution is entirely designed, built and supported by the same expert Microware team.***

Modularity Lets YOU Choose Just What You Need.

The modular design of OS-9 allows our Operating System to adapt as your requirements change. OS-9 can support a complete spectrum of applications — from embedded ROM-based code in board-level products all the way up to large-scale systems.

The OS-9 Success Kit

A Total Integrated Solution for Your Next Project

Development Tools:
- C Source Level Debugger
- Symbolic Debugger
- System State Debugger
- uMACS Text Editor
- Electronic Mail
- Communications
- Super Shell

Kernel Options:
- MMU (Security Protection) Support
- Math Coprocessor Support

Languages:
- C*
- Basic
- Pascal
- Fortran
- Ada**
- Assembler*

I/O Options:
- SCSI, SASI & SMD Disks
- 3-, 5-, 8-inch Diskettes
- Magnetic Tape
- Ethernet - TCP/IP
- Arcnet - OS-9/Net

*Resident or UNIX versions available
**VAX hosted

Support is Part of the Package.

Microware is proudly setting the industry's standard for customer support. You'll find professional and comprehensive technical documentation and a Customer Hotline staffed by courteous and authoritative software engineers.

So stop messing with simple kernels and independent suppliers. Call Microware today and find out more about the "One-Stop Integrated Solution" with OS-9!

microware® OS-9

Microware Systems Corporation
1900 N.W. 114th Street
Des Moines, Iowa 50322
Phone: 515/224-1929

Western Regional Office
4401 Great America Parkway
Santa Clara, California 95054
Phone: 408/980-0201

Microware Japan Ltd.
41-19 Honcho 4-Chome
Funabashi City
Chiba 273, Japan
Phone: 0474 (22) 1747

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	3.1	3.2
68008 OS-9 68K 8 Mhz	18.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.2*	*0.89*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.22*

```
Main()
        {
            register long i;
            for (i=0; i < 999999; ++i);
        }
```

Estimated MIPS - MUSTANG-020 .... 4.5 MIPS,

Burst to 8 . 10 MIPS: Motorola Specs

OS-9

OS-9 Professional Ver *Includes C Compiler	$850.00
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Pascal 77	750.00
Microware Pascal	500.00
Oregon Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PacCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Pascal 77	450.00
Sculptor+ (see below)	995.00
Standard MUSTANG-020™ shipped 12.5 Mhz.	
Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232	335.00
Requires 1 or 2 Adapter Cards below RS232 Adapter	165.00
Each card supports 4 additional ser. ports (total of 36 serial ports supported)	
60 line Parallel I/O card	398.00
Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.	
Prototype Board	75.00
areas for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.	
SBC-AN	475.00
Interface between the system and ARCNET modified token-passing LAN, fiber optics optional - call. LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00
Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. SAVE $300.00	
Software Discounts	

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

Mustang Specifications

12.5 Mhz (optional) 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
buffered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

The PRO!

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a must.

Only the "PRO" Version of OS-9 Supported!

This is *HEAVY DUTY* Country!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Slipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note all 68881 chips work with 20 Mhz Sys		
	Total:	$5299.80

NEW LOWER PRICES

25 Mbyte HD ~~$4299.80~~ $3749.80

85 Mbyte HD ~~$5248.80~~ $4548.80

Data-Comp Division

A Decade of Quality Service"

Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Note: Only Professional OS-9 Now Available (68020 Version) Includes ($500) C Compiler - 68020 & 68881 Supported - For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Publisher
Don Williams Sr.

Executive Editor
Larry Williams

Production Manager
Tom Williams

Office Manager
Joyce Williams

Subscriptions
Cheryl Hodge

Contributing & Associate Editors

Ron Anderson
Ron Voigts
Doug Lurie
Ed Law
Dr. E.M. "Bud" Pass
Art Weller
Dr. Theo Elbert
& Hundreds More of Us

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.
"Over a Decade of Service"
"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343
Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

---

68 Micro Journal (ISSN 0194-5025) is published 12 times a year by Computer Publishing Inc. Second Class Postage paid at Hixson, TN. and additional entries. POSTMASTER: send address changes to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should be flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9 68XXX Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED **SCREEN ORIENTED TEXT EDITOR** with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included**. Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO PAT/JUST

Special $249.00

JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

C User Notes

The C Programmers Reference Source. Always Right On Target!

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter begins with a statement by Dennis Ritchie concerning the ANSI C standards committee. He placed it on USENET, with the usual disclaimer that the opinions expressed are his, and not those of his employer. It is reprinted here verbatim, without editing. For those who do not recognize the name, he is the "R" of "K" and "R".

For the tenth anniversary of the original publication of the K & R book on the C language, there is a new edition, encompassing the new features which have been added to the language in the last ten years. Although it attempts to cover the new ANSI C standard, be aware that the standard is still changing and some features noted in the book have been deleted from the standard. This includes the "noalias" keyword. The "volatile" keyword may also be modified in meaning before the standard is finalized.

Following the statement by Ritchie are commentaries (also from USENET) by others concerning various aspects of the C language. Their arguments show that the C language definition needs additional specification in several areas.

The example program produces text suitable for input to a text processor, such as nro, from plain text not containing the necessary commands.

EVALUATING ANSI C - Dennis Ritchie

Since the subject has come up, this is a good time to record my thoughts on ANSI C and the work that X3J11 has done. In brief, I think the result is commendable. I concur in the belief of those who watch such things that X3J11 managed to improve on what they started with, and that this is an unusual accomplishment for a large committee that lasts for 5 years or so. In particular, they successfully resisted the usually inexorable pressure to add features and options.

The committee had certain explicit goals, among them to bring the language specification up to date with reality, since much had changed in the 10 years since the original definition; to add a very few things deemed necessary (function prototypes); and to specify things more completely than K&R did. They also wanted to standardize the library, something that has become important not only because the drift in the System V vs. BSD worlds, but even more because of the vastly increased use of C on non-Unix systems.

Perhaps as important as all the rest was a goal that was not stated explicitly, namely to supply the stability, legitimacy and general cachet that is possible only with all the legalities required to get the result accepted as an Official Standard by the appropriate Official Bodies. The need for this can be questioned, but the fact that many people want it to happen, and that it tends to occur regardless of desires, can't be denied.

I have only a few worries about the specification of the language.

First, the introduction of the new function declaration mechanism. The new scheme is better than the old, but the change is going to cause trouble, and the need to accommodate both ways confuses the language specification and will confuse users. I made my peace with the change some years ago; it is better to do it than not.

Second, I don't think type qualifiers have been fully digested. I never did object very strongly to volatile, even though I maintain that it is not onerous to live without it. The remaining qualifier, const, suffers from tension between conflicting views of what it is supposed to mean. One possible view is that const things are real constants. If it prevailed, then one would expect that things of type const could appear where constants are expected (case labels and so forth). Perhaps even the notion of pointer to constant would become suspect. This was not the view that the committee eventually adopted, although (perhaps unconscious) sentiment for it remains; instead, a more implementation-minded approach was taken, namely that const means something that can be put in ROM (hardware or write-protected memory). Nevertheless, the tension remains, and figured heavily in the committee's arguments over seemingly unrelated things (including noalias). Even now, questions of whether a pointer to const can be assigned to an ordinary pointer, or whether these two types are compatible (and when compatibility matters) are lively issues. It is clear that the rules on qualifiers are to some extent artificial—they could be decided in several ways. Good rules are not arbitrary, they are forced by the logic of the design.

Some of the things that people are complaining about were completely necessary, in particular the insistence that only a fixed set of library routines is defined, that initial _ is reserved for the implementation, and that all other names are reserved for the user. It is just not possible to write a standard that permits the implementation to intrude into the user's name space (by letting it define names internal to its library that the user might use by accident), or to give a defined, standard way for the user to replace system routines. The latter would constrain implementations too much. At the same time, the consequences of such rules are misunderstood. A program that calls "read" is not defined by the standard, but I assure you that the supplier of your Unix system will arrange that it works.

X3J11 did miss some opportunities. Perhaps the most obvious lack in the language is a scheme for variadic arrays. However, the proposals I have seen are awkward and don't fit smoothly, so it is not surprising that nothing was done.

Another thing that must await the future is a genuine rethinking of integer arithmetic, not just the fiddles that they did.

All in all, I think X3J11 did an excellent job. When they began, several years ago, I was somewhat apprehensive about what would result, but also decided that I did not have the stamina to become involved in their activities; I would have to trust their good sense. I have never regretted the decision, and I'm pleased with the outcome.

OVERLOADING C OPERATORS

The following commentary is due to Frank Byrum of BROOKS Financial Systems, Inc.

They recently experienced an exceptionally insidious error in their production code, which was causing sporadic write failures on execution. This error successfully eluded detection by a DOS compiler as well as by a full linting under Unix System V Release 3.

The intent was to have the following C code:

```
if((lpfd = open(printer_string(PORT), O_WRONLY)) < 0)
        (fatal error message) ... ;
    write(lpfd, ... , ...);
```

In English, if the return value from open()ing the printer as assigned to lpfd is not a valid file descriptor, quit; else use it.

Unfortunately, what was actually coded was the following:

```
if((lpfd = open(printer_string(PORT)), O_WRONLY) < 0)
        (fatal error message) ... ;
    write(lpfd, ... , ...);
```

Take careful notice of the change in parenthesizing. These statements constitute perfectly legal C. Notice first that open() is called with an insufficient number of arguments. The stream's status will be based on a random value taken from the stack! Whether this value implies writability is a random parameter, hence the variable nature of the bug. The comma is now a comma operator, since it is outside the function brackets. Assignment takes precedence over the comma operator, so lpfd is indeed assigned the return value from open(), but this entire assignment now constitutes the left side of a comma expression, so it is the value of O_WRONLY that is compared against 0! O_WRONLY is defined as a positive constant, so the test always fails and attempted execution of the write() becomes inevitable.

The lint program did not report an argument count error for open(). It so happens that open() is one of the few system calls that can take a variable number of arguments (depending in this case on whether the O_CREAT flag is included in the second argument). Hence lint's argument count check is suppressed by the /*VARARGS2*/ modifier in llib-lc.

It took them a very long time to find the problem, as could be imagined.

He noted that C's overloading of certain operators (in this case, the lowly comma) increases the probability that an erroneous code fragment will be legally interpretable, a situation somewhat analogous to having too many words in a spellchecking dictionary. This lead directly to the compiler not locating the logic error, since it was not considered a syntax error.

GOTOS FURTHER DEBATED

The following section was written by William Wells of Proximity Technology of Fort Lauderdale, Florida. As usual, his opinions are his own, not his employer's.

Under ordinary circumstances, there is exactly one place where a human C coder might use a goto. This is to implement multi-level breaks and continues.

I say this, having managed (and written huge chunks of) a 17,000 line software system (and that is only the part we sell, and does not include development tools). I have programmed in C for six years now and have NEVER used a goto. We have uncounted megabytes of C code written in-house. None of it (to my knowledge) contains a goto. The closest thing we have to a goto is setjmp/longjmp, used to implement a multi-level return (and that is a recent change, one whose contemplation caused much debate).

With that aside, let me explain why the goto discussion is really fruitless. People have observed that gotos are used in a lot of bad code. From this it is concluded that gotos are bad. This is really bad logic. Try this: programmers have been observed to write bad code; therefore, programmers are bad!

There is nothing wrong with goto. How do I reconcile with my statements above? Wait and see... The thing that is wrong is the control structures being implemented with the gotos.

The whole point of the structured programming debate is this: every program has a control structure; some of these control structures are better than others. Whether you use gotos or some other language feature to implement the control structure does not change what the control structure is nor does it affect the goodness of the control structure.

The quality of your program is strongly influenced by the quality of its control structures. Furthermore, you want that control structure to be obvious and understandable to the reader of the program. This implies that you use language features that make your use of a control structure obvious.

So, the first question should be: what are the good control structures?

The second question should be: given a particular language, how should the control structures be implemented?

Ok, so what makes a control structure good? Well, the basic answers are: a control structure is good if it is

1) appropriate to solving problems.
2) easy to write.
3) easy to understand.
4) easy to maintain.
5) ... add your own as long as they do not contradict the above.

There are obviously lots of control structures that meet these requirements and you do not have to use all of them. In fact, you should pick a set of those which are most appropriate for your programming environment and use them. This set should be, in some sense, a minimum one; for example, if you have two control structures which can accomplish the same thing, but one is easier to use than the other (for you), pick the easier one and forget the other. All other things being equal, a smaller number of control structures helps make your program easier to understand.

Now, I hope my claim about our C programs is understandable. But if not, here is what it amounts to: I have chosen a set of control structures which is appropriate to programming in C, for the kind of programming tasks that I do. It happens that, while my set of control structures includes multi-level breaks and continues (which would be implemented with a goto), I have never had need of one. Given the amount of code I write, it seems to me that one might never need to use an explicit goto in C code.

Now that I think of it, here is a reason to avoid naked gotos in C code: for all other constructs, the control structure being implemented is obvious from the keywords employed. This is not true for goto. Therefore, supposing that you have found a control structure that you have to implement using gotos in C, you should dress the goto up. As an example, suppose that you are using the state machine control structure.

I normally code it as the following:

```
state = STATE_INIT;
    while (state != STATE_DONE)
    {
        switch (state)
        {
        case STATE_INIT:
        }
    }
```

However, this is not the most efficient way to do it. You could also implement it as the following:

```
/* Wherever you see the macros state and nextstate being used,
you will be seeing a state machine. The state macro defines the
start of a state. The nextstate macro causes a transfer of control
to another state of the same machine. A state machine starts at a
6efine of statepref and ends with state(DONE). */

6efine dummy(x)        x
    6efine state(x)          dummy(statepref)x
    6efine nextstate(x)      goto dummy(statepref)x

    4fdef statepref
    0ndef statepref
    1ndif
    6efine statepref         STATE_

        state(INIT):
            (code for this state)
            nextstate(DONE);
        (more states with appropriate code)
        state(DONE):
        (code after state machine)
```

I am aware that not all preprocessors will do what I want here; for real portability, you would explicitly write the prefixes. Also, this method fails for nested state machines, something I have occasionally had need of.

Some of you will no doubt be thinking: but why should I go to all this effort when I could just use the goto directly? Well, if this was all you did with goto, I don't really see any reason why not (but I do think your program should include a comment saying that you use goto for state machines and describes how you structure it). If, however, you have more than one way of using goto, you should clothe the gotos somehow so that the reader of the program knows what control structure your goto belongs to. After all, a while is just a disguised goto.

WHITE-SPACE IN C SYNTAX

Almost without exception, white-space may be included or excluded from C programs without affecting the logic of the program. The obvious exceptions are within quoted strings, identifiers, keywords, numbers, and multi-character symbols, where white-space may not be included without splitting one unit into two.

A problem in some of the earliest C compilers, which is described in the original K & R book as being an anachronism, was the older form of the binary-operator/assignment statement. Rather than being coded as i -= 1, it was coded as i =- 1, leading to the possibility of an ambiguity if the spaces were removed, since i=-1 could represent a binary-operator/assignment statement or an assignment statement. Unfortunately, some of the current C compilers accept this syntax, either silently or with warnings.

Certain implementations of the C language, especially older ones, had dependencies and peculiarities which made white-space significant in unexpected contexts.

For example, one C compiler required no white-space between an array name and the open bracket introducing the first array dimension. This situation generated only syntax errors and some grumbling on the part of the programmers attempting to use the compiler.

Another C compiler required no white-space between a function name and the open parenthesis introducing the function arguments. Again, this was primarily a nuisance.

The same C compiler required no white-space between a parametric define name and the open parenthesis introducing its arguments. Unfortunately, if it found no adjacent open parenthesis, it assumed that the arguments were all null and expanded the define. Then, it encountered the open parenthesis and evaluated the arguments of the define as expressions separated by comma operators. If the programmer were lucky, the compiler generated a syntax error. However, in many cases, the compiler silently generated incorrect code because it mis-interpreted the statement.

C compilers conforming to the ANSI C standard must treat comments syntactically as white-space. Many older compilers, and several current ones, treat comments as if they were not present. The difference is probably academic, but may cause minor portability difficulties.

There is one ambiguity in the C language definition which has been present since essentially the beginning and continues into the ANSI C standard. If a divide operator is immediately followed by the asterisk of a dereferencing pointer operation, the result will be indistinguishable from the "/*" multi-character symbol introducing a comment. This will cause a problem in any C program containing this sequence, but it will be exceptionally severe with current C compilers which do not support nested comments, as the resulting comment will encompass all code up to and including the next comment.

The easiest manner in which to avoid most ambiguity is to use white-space liberally. This assists both the human and the computer in interpreting the program correctly.

C PROBLEM

Following is a puzzle on unsigned promotions, posed by Steve Friedl of V-Systems, Inc. It illustrates a portability issue with sign extension.

Can any of you help with a puzzle? I've been trying to understand the unsigned vs value perserving rules of various C compilers, and I'm afraid I've run into a case I don't understand. I have pored over the dpANS on this but am still confused.

Note that this question applies to all compilers, not just dsANSI ones.

```
/* what is printed? */
    6efine          MINUS_ONE     0xffff

    main()
    {
        unaigned ahort     aval;
        long lval1, lval2;

        aval = MINUS_ONE;

        lval1 = - aval;
        lval2 = - (unaigned ahort) MINUS_ONE;

      printf("lval 1 = 71d    2 = 71dn", lval1, lval2);
    }

The normal answer I get is:

    lval 1 = -65535     2 = -65535

but on the HP9000 I see:

    lval 1 = 1     2 = -65535
```

For what it's worth, the HP9000 has sizeof(short) = 2 and sizeof(int) = 4, and I get the same results when I 6efine MINUS_ONE to be (-1) or 0xffff.

My specific questions:

(1) is this a case of questionable unsigned-ness?

(2) if I have a vendor who asserts "this is a value-preserving compiler", what is the necessary value of "lval"?

(3) if I have a vendor who asserts "this is a unsigned-preserving compiler", what is the necessary value of "lval"?

(4) if I have a vendor who asserts "this is a dpANS-conformant" compiler", what is the necessary value of "lval"?

EXAMPLE C PROGRAM

Following is this month's example C program; it converts text files to a simple nro format. It assumes that text begins in the first column and that non-text (tables, etc.) begins in other than the first column. By changing the commands somewhat, the program could be modified to output commands for another text processor.

```
#include <stdio.h>
#include <ctype.h>

char *p;
char string[256];
int i;
int n;
int s;
int t;

main(argc, argv)
int argc;
char *argv[];
{
    while (fgets(string, 256, stdin))
    {
        if (*string > ' ')
        {
            if (i)
                i = 0, fputs(".fin~", stdout);
            if (t)
                t = 0, fputs(".in 0n~", stdout);
            if (s)
                printf(".sp 8dn~", s), s = 0;
            fputs(string, stdout);
        }
        else
        {
            for (p = string, n = 0;
                isspace(*p); ++p, ++n);
            if (*p)
            {
                if (!i)
                    i = 1, fputs(".nfn~", stdout);
                if (t != n)
                    printf(".in 8dn~", t = n);
                if (s)
                    printf(".sp 8dn~", s), s = 0;
                fputs(p, stdout);
            }
            else
                ++s;
        }
    }
}
```

+++

SOFTWARE USER NOTES

A Tutorial Series

By : Ronald W Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

Centcolmnal

This is my 100th column, though not all 100 have appeared in 68 Micro Journal. Way back when, I started out to publish a newsletter, advertised in "Kilobaud Microcomputing" via letter. I had printed, duplicated and distributed 8 newsletters when Don Williams phoned one evening to ask if I would like to switch to writing a column for '68' Micro. I said that would be fine, but I had committed myself to 12 newsletters, and had collected a small fee that would just about cover cost of duplication and mailing. Don said that he would send four issues of '68' to my list of about 25 initial subscribers to fulfill my obligation. Of course, my "column" was then read by a few thousand hobbyists rather than just 25 of them. That means that I have written some 92 columns for 68 Micro Journal, and I've missed more or less 4 times (including two recently). That means that it has been just about 8 years that I've been here in this spot.

My early columns included a lot of software listings, mostly utilities for the 6800 FLEX09 operating system. When the 6809 came along, I did the same for that. I had little choice, since there really wasn't much to talk about in software available for those early microcomputers. Technical Systems Consultants had an ad very early in Kilobaud, selling some little games that would run on a KIM-1 board with terminal attached. The games were fun but truly primitive by today's standards. There was HANGMAN, of course, and REVERSE in which you tried to order ten digits that had been randomized by reversing some fraction of them from a given position on to the end. The KIM-1 was a 6502 based single board computer with 256 BYTES of RAM. Before the addition of extra memory and a terminal it ran little games on its 6 seven segment LED displays. It had a keypad with which you could explore memory or enter programs a byte at a time. Programs could be saved to audio tape and loaded from it as well. There was a KIM-1 newsletter in which were published little games, such as a horse race in which the three horizontal bars of the seven segment displays were illuminated in turn advancing from left to right based on a random number generator. The first one to make it across the 6 digits was, of course, the winning horse.

Eventually I found a Tiny Basic for it, and attached a 4K memory board, adapting its S-100 bus to the KIM bus.

About January 1977 I ordered my Southwest Technical Products 6800 box and a terminal that consisted of a keyboard and a logic circuit board that included a character generator etc. I attached an old television set and the combination could be used as a 40 column terminal! SWTPC used a cassette as the software medium at a whopping 300 baud transfer rate. I remember starting "16K BASIC" loading to the computer and breaking for supper. When I came back either BASIC was running or I had an error message and had to start over again. TSC offered their games in 6800 version and I bought those again. Then they offered an assembler and a line editor. I bought both (to save money) in source code form and typed in the HEX DUMP to get them running. That was an exercise in frustration, searching for a wrong byte here and there in the code until it all worked correctly. Later I typed in the source code and got it assembled. SWTPC had a printer to go with the computer. You guessed it, it

printed 40 column on roughly 4 inch wide "adding machine" tape to match the 40 column display.

A year or so later, I bought a genuine ADM-3A terminal (with the option to allow both upper and lower case), and then a pair of 5 1/4 inch disk drives which came with a primitive disk operating system (that worked fine, I might add). Six months down the road from then, SWTPC sent all the customers who had bought floppy disk systems a copy of Miniflex, the first REAL operating system for the 68XX processors. I remember that they had stuck the disk in the middle of their printed manual and sent it in a 9 by 12 envelope which the mail man nicely folded in half the long way, to fit the mailbox. I improvised and found that I could remove the magnetic medium from the disk jacket by cutting one edge of the jacket open. Then I did the same with a good jacket and moved the insides to that. I could then read the disk and copy it to a good disk and I was running Miniflex. TSC offered their line editor EDIT and their assembler ASMB again, and this time I bought them on disk. TSC supplied printed source code. They did the same for their text processor called simply PR. I bought that as well. There was a very early compiler by Jack Hemmenway called Strubal (STRUctured BAsic Language). I can't say much more for it than to say that it worked. I typed in a 6 or 7 page program and the compiler output overflowed memory! (I had 32K at the time). Lucidata released their 6800 Pascal Pcode compiler that was many times more efficient at code generation... (Not a knock for Hemmenway, just progress happening).

Shortly, TSC came along with their DEBUG program that let the user single step through a program, set break points, etc. I wondered at the time how I had ever debugged an assembler program without it.

A year or two after that I upgraded by buying a 6809 processor board and a lot more memory for the system. It originally had 16K but I had previously upgraded to 32K and now I went to 48K and then 56K (the maximum possible in that computer system). FLEX09 came along for the 6809 and went through a few upgrades. Then started the era of fast and furious development in the area of software. Lots of software was developed by hobbyists and then sold through '68' Micro Journal and other sources. If I am correct, only Technical Systems Consultants and Percom, and perhaps SWTPC itself supplied software that was specifically written with profit in mind, at least early on.

There was TSC's original Text Editor and Processor combination (updated to 6809 FLEX version), their Assembler, Lucidata's Pascal compiler, followed by a couple of "C" compilers. The first, from a little company that will remain nameless, was a disaster. The author was not a great programmer, but worse the company hadn't the vaguest idea how to test software. There were so many bugs that a program compiled under it seldom ran correctly. Later Windrush Microsystems in England released a "C" compiler by James McCosh that was as good as the earlier one was bad. McCosh is a good programmer, and Windrush knows how to test software and respond to bugs reported by users. I still use that "C" now and then when I fire up the 6809 system. It is, in fact a direct ancestor of Microware's present OS9 68K "C" compiler. The folks from Microware might say that it has come a long way since then, and I'd agree heartily to that. Other software that came from that era was the PL9 compiler, Stylo Text Editor, OmegaSoft Pascal, PIE text editor, a couple of implementations of FORTH, and much more. About the end of the 6809 era I received a COBOL compiler for evaluation. Just about every new piece of software that came along (most or all of them were written in Assembler) showed a greater understanding of how to program a 6809. The compilers each generated better and smaller code than their predecessors. One peculiar language came along that was very good at doing things I don't normally do in my programs and terrible at doing things that I usually do. I let someone else review it and it got a fair write up.

Then, along came IBM with their PC and PC-XT and the big switch began. Many of the 6809 users remained loyal and we wrote much of our own software, but it became evident that there would be no more commercial software available for these systems. I wrote PAT, an editor which was about a year's worth of spare time effort. I figure I've probably cleared about a dollar an hour for that effort, but it wasn't done to get rich. I really did it to prove that good software could be written in a higher level language (PL9), and that it could be sufficiently small to leave a reasonable edit buffer, and run fast enough to be adequately usable. I did end up coding about 3% of it in Assembler as

ASMPROCs in order to speed up some very processor intensive operations such as string searches.

(I am at the point now with the 68000 systems where I was back about a year after the 6809 came along. Software is limited but coming. I've managed to get enough together to do whatever I want to do, and I've filled in with some "homemade" utilities in assembler, all of which have been published here at various times).

Somewhere along the era of all the new software arriving for the 6809, I began to slant the column more toward programming in higher level languages, and in reviewing the various offerings from the suppliers. I got in deep trouble with a few readers by saying once that I would choke (or worse) if I saw yet another COPY or DIRectory utility for the 6809 published. My comments by pure coincidence appeared in the column just about the time a reader sent me a disk with some nice things on it, but including a directory utility. The writer took personal offense and didn't believe that my comments were not aimed at him specifically! I got a very nasty letter and wrote a couple letters of explanation which must have been transferred directly to the garbage, because there was never another response. (About then I thought seriously about quitting this column). With regard to the comment about Copy or Directory utilities, what I had in mind was that computers had to serve some useful purpose beyond allowing their users to create more utility programs for the operating system. One "hacker" (I don't use the term in any sort of derogatory sense) wrote and told me that he didn't intend ever to do anything useful with his computer! I've since adopted a similar attitude toward the more typical computer uses. I told someone that I would quit my job if I ever had to work with or on Database software or a Spreadsheet. I do appreciate the value of such software and computer uses, but they are not for me. I am an Electrical Engineer by education, and I immediately saw applications in the field of instrumentation and control of machines. It is in that area where my interests lie. So much for history except to note that the name of this file is NOTES100. Since 1000 months is 83 years and 4 months, I'm sure I won't have to shorten the name after the 999th column!

As you may have noticed, I'm back into hacking again, writing new utilities for the 68000 and SK*DOS. Should any of you think that after ten or twelve years, it must be a total breeze for someone to write programs, let me tell you about last night. A couple of days ago I decided that a rewrite of my text processor JUST was long overdue. I first wanted to set it up to read a printer configuration file so that the text processor software cold be fixed and constant. The user could then prepare his own printer configuration file using only printable characters to represent the control sequences. That turned out to be a relatively easy one-evening plus, project. Then I decided to try to clean up some of the procedures, which had "grown" from the initial "keep it simple" concept to have features added and more features added. I was successful in getting it to correctly center and right justify a line in double wide printing mode, and decided that it would be nice to fix it so it could justify a line with double wide printing in it also.

Having done similar things before, I made a careful backup of what I had that worked correctly, and set off to make the necessary changes. That was about 9:00 in the evening. Things went from not working to still not working to not working as well as before, to total garbage. At one point, I actually was printing out every other letter in the line with a blank space between! Finally about 1:30 AM, I conceded defeat and threw away my new effort and reloaded my backup. In the four hours of effort, I had found and fixed one bug, so I added the fix to the original file. I guess I'll stare at a listing and let that project sit for a while. The time was not wasted, since I now have gotten back into the program and have a much better feel for how I had done it several years ago. A rewrite will be based on what I learned. My procedures were all much too interdependent. I'd change one line and something that ought to be totally unrelated would no longer function properly.

When I was a lot younger and living in Chicago, I had a friend named Dick Neilsen. Dick always had a "shortcut" to get somewhere. Taking Dick's shortcut usually resulted in getting lost and taking twice as long to reach the destination. Among Dick's friends we began referring to doing something the hard way as a "Dick Neilsen Shortcut". Sometimes a poorly thought out shortcut in trying to write a program is just that, i.e. doing it the hard way. When that happens, it is usually best to think about the problem a little more and then try

again. I had a few good ideas and a couple of nights later I started again and made at least some progress. I now have a couple of new versions of JUST both working and doing more than the old one, but they are lacking in some of the features that I want to add.

I am obviously too much of the "start programming and get something working, and then clean it up later", school. I think some programmers go too far in the direction of getting it absolutely right on paper and expecting it to work on the first try. I've said that there is a middle ground that should result in minimum overall programming and debug time, and I observe that when I spend about equal amounts of time programming and debugging, I seem to get the job done fastest. I maintain that if I were to spend three times as long writing the program, I would have about 1/3 as many bugs, so that the program plus debug time would be about 3 1/3 units. Equal program and debug time would add up to 2 time units. Similarly, 1/2 the programming time would result in 2 times as much debug time, so that the minimum is about where the program and debug time are equal. I realize that this is not something that can be derived mathematically nor proven rigorously with logic propositions, (nor for that matter anticipated before the debug is done) but it would seem to be verifiable by experience. If I find that I spent too long debugging the last program, I spend more time writing the next, etc.

Based on my experience of the other night, I conclude that I didn't understand the problem thoroughly enough to be able to program well. In many instances when I write a program, I don't know exactly what it has to do until I take a good stab at it and try it. Then what it has to do sometimes becomes very obvious, and turns out to be significantly different from the original concept. This is particularly true in the area of machine control and instrumentation, though I am sure it carries into other programming areas as well. If the problem is REALLY spelled out well, that is a different case, but I don't believe that happens in practice nearly as often as the theorists would have us think.

Let me give you an example of this. I have been working on a word processor again, so I'll use that. How do we read text from a file with ragged right margin, not necessarily with every line filled as full as possible, and print it out fully justified, i.e. each line the same length (by inserting spaces in the line). A first approach might be to read characters from the input file until you have LENGTH characters. If you come across a CR change it to a space. Now back up through the partial word at the end of the line (if it is partial) and past the space that separates it from the previous word. Of course you count the number of places backed up, and then add that many spaces within the line. You put a CR at the end of the line and then print it. (How you distribute the extra spaces is another problem we'll assume you have solved). Now move the partial word left over from that line to the start of the next line and repeat the process until end of file.

If you do that, you will, of course have run the whole file together into one big paragraph. Titles and headers are not part of a paragraph. Of course you must have a way to tell when a paragraph has started and only treat the text within a paragraph as described above. Some authors of text processors choose to make a paragraph one long line. That is, the editor is set up to handle formatting on the screen without any CRs in the text. That simplifies the identification of a paragraph, but the file is hard to list using an ordinary list utility. I chose to use a signal inserted in the text to mark the start of a paragraph. ".p" all by itself on a line signals that a paragraph starts on the next line. The end of a paragraph is detected by two consecutive CRs (a blank line after the end of the paragraph) or another .P signaling the start of a new paragraph. When the paragraph is ended, a partial line might remain in the line buffer. Of course it has to be printed too, and it might as well not have extra spaces in it since it will probably not be a full line anyway.

Do you see what I am getting at? Unless you are an absolute genius (or at least a lot smarter than I), you won't think of all the conditions and exceptions without making a stab at a way to solve the problem and trying it to see what happens. Trying to write a program that is correct the first try is like trying to think up all the conditions and exceptions without trying what you have done so far to see if it works. More experienced programmers at least think of how a process like that described in the previous paragraph must start and stop, and perhaps they catch a few

more of the exceptions, but to try to catch them all without "experimenting" would seem to me to be a great waste of time.

Epson "Enhancement"?

A few months ago I saw an add in Computer Shopper for a new ROM to upgrade an Epson MX-80 among other models. I have a very old one that still works like new. I had upgraded it a long time ago with GRAFTRAX, the Epson upgrade, but of course it still didn't have "NLQ" mode. The ad for an upgrade kit called "Dots-Perfect" indicated that the upgrade would add letter quality and control of printing mode via the Online, Formfeed, and linefeed buttons, more or less like the newer Epsons.

I sent for the upgrade and it has arrived. I followed the instructions and it all worked pretty much as advertised on the first try, but the "NLQ" looks pretty bad, in fact not as good as just printing in Emphasized mode as I had been doing with that printer ever since I bought it. Certainly it doesn't compare with the LX-800 in either of its NLQ modes in which the characters look very full and smooth even under a magnifying glass. Vertical lines are quite obviously zigzag. It appears that the NLQ is achieved by simply shifting the carriage and overstriking.

The "upgrade" cost $69. In my opinion, $20 would be more in line with what it does, primarily based on the poor NLQ. The Epson MX-80 is capable of the same graphics resolution as the new LX-800, so there is no reason that a better NLQ font couldn't be done, other than, perhaps, lack of ROM space for the font character definitions. The upgrade does allow selection of fonts and print styles using the On Line, Formfeed and Linefeed buttons as advertised, and those are also selectable via software compatible with the latest Epson LX-800. It even has a "small print" mode that is simply superscript mode, condensed mode, and half line feed all combined. That's nothing you couldn't do on any Epson with true half size superscripts and subscripts under software control. I suppose if I didn't know how to write or configure word processing software, I'd be more impressed.

In would think you'd be better off spending that $70 as a significant fraction of the cost of a new Epson LX-800 or even a Citizen 180-D which can be had for around $170. It is a nice little printer but its NLQ is a little thin compared to the LX-800.

CRASH!

I just found a way to crash my editor PAT. I'll mention it so you can avoid it if you are a PAT user. I accidentally typed ESC ^L. Of course ESC 65 ^L would set the line length to 65, but with no parameter the line length was set to 0. I didn't notice that, and tried to format a paragraph to zero width. The "BUSY" notice came on, and stayed there until I ran out of memory, at which point there was an error message and an exit to the monitor. Of course all my edit was lost. I'll trap the no-parameter case and make it not change the current line length. Meanwhile, PAT users be warned.

+++

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street
Leominister, MA 01453

OF-Link from Sardis Technologies

At last, a way to combine FLEX and OS-9 Level II! Sardis Technologies has a real winner, here. If you have been reluctant to switch to the Color Computer and OS-9 because of a heavy investment in FLEX software, OF-Link could be the answer to your prayers. Now you can have all of the advantages of OS-9 without losing that hoard of FLEX software you have accumulated over the years.

I don't want to spend any time going over the virtues of OS-9 vis-a-vis FLEX, as I am sure that you have heard enough to be tired of that argument. Rather, I want to describe my reaction to running FLEX programs in this new environment.

First, let me say that not every program will work. I could not find any software in my admittedly small inventory of "regular" FLEX goodies which would not run completely as expected under OF-Link. However, the copy of STYLOGRAPH which I use is the one especially tailored to the CoCo FLEX sold by South East Media, and I was not able to finagle the patches to make it run under OF-Link. Similar troubles would arise with any software which does not use the standard FLEX hooks. The instructions which come with OF-Link give help with some popular software which has a problem, but this is still for "regular" FLEX programs.

There are stringent hardware requirements, also. You must have a CoCo3 with 512K of RAM and two disk drives; otherwise, you cannot install the system. However, once OF-Link has been installed, you then can manage with only one drive, although it will be inconvenient.

The reason that you MUST have two drives is that you must install D. P. Johnson's SDISK3 before you install OF-Link. Contact D.P. Johnson at 7655 SW Cedarcrest St..Portland, Oregon 97223, (503) 244-8152. *See D.P. Johnson's ad on page 59.* If you have a DMC "no halt" disk controller from Sardis, you have no problems, since you already have SDISK3. If, like me, you have a Tandy, Disto, or other controller, then you must separately obtain and install SDISK3. I followed this route with both the original Tandy controller and the Disto SCII, and built a working system on SSDD35 drives, so you can, too. Just follow the excellent directions exactly and literally, and don't free-lance!

Once you have a working system, you can use modpatch or the public domain dmode to change to any number of tracks you wish. I easily converted from SSDD35 to SSDD80 by way of dmode. Unfortunately, I could not get any sort of double-sided drives to work; this may have been the result of conflicts with my particular hardware. Also, I could not get more than 2 physical drives to work, even though the manual gave me the impression that I could. However, I had no problem with as many as 4 logical drives, so this may not be a practical limitation.

There is one "gotcha" which comes from the peculiarities of OS-9 record locking, and not from OF-Link. In any case, each active FLEX device window must have its own virtual system file. In other words, if you have 3 windows running with FF9, you must have 3 separate virtual system files, one to support each device window. I solved this "problem" by having all but one of the virtual windows contain only FF9 and the absolute minimum of other files, such as FLEX.COR, etc. Only one of the virtual system files contains a full complement of FLEX utilities. This saves physical disk space. By naming the 3 files (remember that these are really OS-9 files in disguise) as fv1, fv2, and fv3. I can call each one as I setup /w1, /w2, and /w3. There is a way around this problem of record locking and it is described in the manual; but the technique I described above is the one recommended as being a lot safer and saner!

Be very careful when you start sharing FLEX files, since it is very easy to mess up the links between sectors and the free chain. If you should have this sort of accident, you are in for a real hassle recovering from it. Be warned!!!

Utilities, etc.

Several very useful utilities are included with OF-Link which make it easy to install and use. The ones I used the most, and I expect everybody will have about the same experience, are dpoke and newcrc. Dpoke lets you change an OS-9 file and newcrc updates the crc without the necessity of explicitly moving that file from and back to the disk.

There is also a simple monitor which will help in debugging FLEX programs. Unfortunately, this was not fully operational on the review-copy of OF-Link which I received, so I can't say more than that it certainly looks good.

FF9 under OF-Link

So much for the preliminaries, now for the important stuff!!

I think that the first question answered should be, "Why bother?" Well, that is the easiest question to answer. With OF-Link, you gain:

1. 80-column screen
2. windows
3. graphics
4. color

Ok, if OF-Link is so good, what do you lose? The only lost feature of FF9 that I have been able to identify, so far, is the direct interface to the machine. However, this is a characteristic of OS-9, and not solely of OF-Link. If you use your FLEX software for machine control, then you must either choose between the great conveniences of OF-Link and the necessity for writing OS-9 device drivers or go back to your old system. No, I have not yet written a successful device driver from scratch—something I have to learn. It may be possible to use the OS-9 call F$MAPBLK, as is done in FORTH09, but I have not worked that out, either.

The 80-column screen is a bigger improvement over the 51-column screen than I had ever expected. As of now, I have put aside my SS50 and separate terminal in favor of the CoCo3. This, in my not-so-humble opinion, alone is worth the cost of the software! I may resurrect the SS50 at some later date, but I can't imagine when or why that would be.

Windows let me have several copies of FF9 going at once, so that I can flick from one program to another just by pressing the <CLEAR> key. Again, I am sure that you have been inundated with praise for this feature of OS-9, so I won't spend time on the hard sell.

By graphics, I don't mean just the crude cartoons to the excellent art that is possible, but also the full business and engineering-style graphics that can be generated by the lowly EMIT .

Color is available, as expected. The options and limitations normally available under RSDOS or OS-9 are all there. If you have the hardware, you can access them.

Melding FF9 to OF-Link

OF-Link gives you the option of using the normal hardware FLEX disks or the virtual FLEX disks which actually are resident as OS-9 files; FLEX just thinks that they are the usual disk.

For use under OF-Link, the virtual disk is to be preferred. This is particularly true if you have a hard disk; you can easily partition it into FLEX and OS-9 segments. OS-9 would think that there were only OS-9 files present, and all of the OS-9 file utilities would work. However, don't try to use a FLEX file utility on one of the OS-9 virtual FLEX files; this is fatal! You can boot FLEX from the virtual disk on the hard disk, so you don't have to keep a floppy around just for that.

The virtual FLEX disk has 64 sectors per track, so you must modify the FF9 disk parameters accordingly. Assuming that you have formatted your virtual FLEX disk to have 24 tracks (to produce 368 screens), you would modify FF9 by entering:

```
24  `  TRK/DRV  >BODY  !
64  `  SEC/TRK  >BODY  !
 1  `  BASETRK  >BODY  !
```

According to my calculations, a single virtual disk could have as much as 2032 screens, if the disk is large enough. Of course, a physical disk can have more than one virtual disk!

Summary

Do I like OF-Link? You bet I do!! Do I recommend OF-Link? You bet I do!!

Sources:
OF-Link, $49.00 (+$5.00 S&H)
Sardis Technologies, 2261 East 11th. Ave., Vancouver, B.C., Canada V5N 1Z7

SDISK3, $29.95 (+$1.75 S&H) D.P.Johnson, 7655 S.W. Cedarcrest St., Portland, OR 97223

FORTH09 ver. 1.02

Dan Johnson has addressed my complaint about FORTH09 shadow screens which I expressed in my review last time. He now allows a command line option "-n" which eliminates any call to shadow screens, so you are not forced to provide for them. Thanks, Dan, I appreciate that!

FORTH09 has some quirks which take a bit of getting used to. These quirks are forced by OS-9 and are not something to be changed on a whim. The most obvious difference with a FORTH operating under OS-9, as opposed to another environment, is that ALL absolute addresses are illegal. Therefore, you cannot legally use jump vector tables.

My observation that the form shown in Screen #3 of my last column would work resulted from a peculiar set of circumstances. This will only work if you compile the screen each time you use it; you cannot run from a SAVEd or a SAVESYSed binary file, since the absolute addresses would then be wrong! Dan also said that this would not work under Level I, so I did not bother to try it.

The proper way to do vectored execution in FORTH09 is through the MAYBE ... THEN AGAIN ... MAYBE NOT construct (I love Dan's choice of names). A workable definition is

```
: EXPERIMENT2 ( n — )
        MAYBE 0 EX0 1 EX1 2 EX2 3 EX3 MAYBE NOT ;
```

This definition will accomplish the same result as the set of definitions and commands I had shown in Screen #3.

Remember, the method shown in Screen #3 is still valid for other environments, just not for OS-9.

OS-9 access from FORTH09

Here is another example of how much I still have to learn about OS-9. I had read all the way through (tough going) the manual which came with my CoCo OS-9 Level II, but the significance of the few lines on manipulating the MMU did not register through all of the other noise. However, the documentation on FORTH09 finally woke me up.

One can easily make OS-9 calls directly from FORTH09, and Dan has provided most of the words one could ever expect to use. Furthermore, he has provided enough information so that definitions can be quickly written to cover the obscure calls that he did not bother with.

Among these calls are F$ALLRAM , F$MAPBLK , and F$CLRBLK , which can be used to access directly any RAM within reach of the MMU. These words can be used to access any RAM block, so that FORTH09 is limited only by the available memory. I wonder if this kind of thing could be used to control a latch which bank-switched additional chunks of RAM, so that the CoCo3 could go even beyond the present 512K? I don't plan to try it, but I am curious.

Another way to access OS-9 is illustrated in the definitions contained in Screens 46-48. Actually, several features of FORTH09 are illustrated in these three screens.

I won't spend much time on the first two screens, since they were discussed in some detail in September, 1987. The only really important points, here, are in the use of VAR , instead of VARIABLE , to define ARGC (line 3 of screen 46). Line 5 of screen 46 and line 7 of screen 48 are identical in showing how to place a value into a VAR . There is no FORTH need to use the loading command twice, but these screens come from my new FORTH09 utility file, so that screen 46 can be used with a reasonable default value in ARGC , when it is loaded without screen 48. On the other hand, ARGC is loaded in the definition of HARD just in case I might have used ARGC somewhere else, so that the value got changed. This is simply cheap insurance against an annoying crash!

Screen 48 shows the use of CR" and SHELL , two words which are unique to FORTH09. CR" places the address of the CR-terminated string on the Data Stack. The string literal must end with a " . SHELL then uses this address to pass the string along to the OS-9 shell for execution and then return control to FORTH09. As you can see, this causes the current date and time to be printed along with the screen listing.

Hardware control through FORTH09 under Level II

I may well be the only one in town who did not realize that, with FORTH09, I could directly access the hardware located at physical addresses $FF00-$FFFF. In other words, this command line:

```
fload forth -n #48k
```

gives me 8K, starting at logical $C000 to play with through F$ALLRAM , etc. and lets me write directly to the hardware I/O, such as the casette motor relay, or anything else located at the physical addresses $E000-$FFFF.

I tried to do this hardware access from BASIC09, but I could not find a way to make it work in Level II. Undoubtedly, it must be illegal, since it is so desirable and easy! In any case, the whole thing is perfectly straight forward in FORTH09. For example:

```
: MOTORON  4  65313  C! ;
: MOTOROFF  52  65313  C! ;
```

Speaking of timing tests, I hope to include some benchmark results with FORTH09 in the near future. I am impressed with the speed of most of the operations; most of the ones that I have checked to date are faster than can be explained by the simple increase in processor speed!

```
Screen 46                                   October 22, 1988  19:19:30
\ ARGC  HELP-SCREEN                                        RDL102088 |0
\ A part of PARSE-COMMAND-LINE                                       |1
                                                                     |2
VAR  ARGC               \  expected number of input parameters       |3
                                                                     |4
2  TO  ARGC                                                          |5
                                                                     |6
: HELP-SCREEN   ( - )                                 \ RDL062587 |7
    CR  CR                                                           |8
    ." Two input parameters are required:"   CR                      |9
    ."   1.  first-scr#"    CR                                       |10
    ."   2.  last-scr#"    CR  CR                                   |11
    ." In no case, can the value of "    ASCII  "  EMIT              |12
    ." first-scr#"    ASCII  "    EMIT    CR                         |13
    ." be greater than the value of "    ASCII  "    EMIT           |14
    ." last-scr#."    ASCII  "    EMIT    CR ;                       |15


Screen 47                                   October 22, 1988  19:19:34
\ ARGV-OK  PARSE-COMMAND-LINE                              RDL102088 |0
\ Requires previous screen                                           |1
                                                                     |2
: ARGV-OK   ( n1 n2 - )                               \ RDL102088 |3
    2DUP                          \ duplicate the input parameters|4
    SWAP    <   \ first-scr# cannot be greater than last-scr#      |5
      IF    HELP-SCREEN                                            |6
        ABORT                                                      |7
      THEN ;                                                       |8
                                                                   |9
: PARSE-COMMAND-LINE    ( - )                         \ RDL102088 |10
    DEPTH     ARGC    <          \ minimum argument count?         |11
      IF    HELP-SCREEN          \ wrong number of arguments       |12
        ABORT                    \ start over                      |13
      ELSE  ARGV-OK              \ verify argument validity        |14
      THEN ;                                                       |15


Screen 48                                   October 22, 1988  19:19:37
\ SHOW                                                     RDL102288 |0
                                                                     |1
SECONDARY  DECIMAL                                                   |2
                                                                     |3
46  47  THRU                                                        |4
                                                                     |5
: HARD ( first-scr# last-scr# - )                     \ RDL102288 |6
    2  TO  ARGC    PARSE-COMMAND-LINE                              |7
    >PRINTER                                                       |8
    1+  SWAP  DO                                                   |9
      CR  ." Screen "  I  .                                        |10
      30  SPACES  CR" DATE T"  SHELL                               |11
      I  LIST                                                      |12
      CR  CR                                                       |13
    LOOP                                                           |14
    >SCREEN ;                                                      |15
```

+++

Logically Speaking

Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

The Mathematical Design of Digital Control Circuits

By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

No TEST last time, ergo no solutions either!

Mile 20 - heading for Mile 21.

Last time, we got introduced to Boolean matrices, and learned how to interpret a matrix to let us trace paths through its corresponding circuit, and we also met Pivotal Condensation, a technique which allowed us to derive a transmission function from a Boolean matrix. Now we'll learn the second of the two techniques I promised, which has certain advantages and also some disadvantages compared to the first.

LAPLACE'S DEVELOPMENT

Our second method, known as Laplace's Development, involves the reduction of our matrix-a to a final 3-by-3, or several 3-by-3s, which, for the benefit of those of you who are familiar with matrix and determinant theory in advanced algebra, will then be evaluated in the same way as a third-order determinant. Except that all "-" signs will be replaced by "+" signs, as there is no such thing as a minus sign in logic theory, at least not at the level at which we're using it. For those of you who've never even HEARD of a determinant, I'll explain everything that it's necessary for you to know in order to work with a third-order Boolean determinant, which is nothing more terrifying than an end-result 3-by-3 matrix to all intents and purposes.

For our first example, we'll begin with our matrix-a of Diagram 102, from which, as you'll see, we'll derive the function f_{12} in no time at all. For 4-by-4 matrices, Laplace's Development gives a very quick solution indeed, but tends to become increasingly cumbersome as we proceed to matrices of higher order, for which I'd recommend using Pivotal Condensation.

DERIVING A DETERMINANT FROM A FOURTH-ORDER MATRIX

You'll recall that in our first method we eliminated all nodes, commencing with the highest numbered node, until we ended up with only those of interest, namely nodes 1 and 2. Strange as it may seem, our new technique seems to do just the opposite. That is, as we're interested in the function f_{12}, WE BEGIN BY ELIMINATING ROW-1 AND COLUMN-2 ENTIRELY from matrix-a. Actually we'll just loop them in red, but, for all practical purposes we've scratched them out. Bang! They don't exist any more! How about that for a quick vanishing act? Then we write down all that remains, as in Diagram 104, and we have our third-order determinant. Note that there's no row-1 or column-2! Because of this fact, our determinant is strictly for f_{12}.

EVALUATING A THIRD-ORDER DETERMINANT

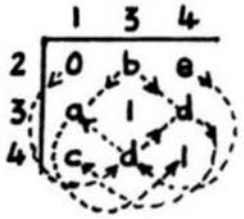


Diagram 104

The best way for a newcomer to "evaluate" such a determinant is to imagine a duplicate copy of 104 written immediately to its right (why don't you do just that?), and then to follow my explanation step-by-step. The dotted lines are merely to show how a more experienced "reader" would achieve the same result without the need of the duplicate copy alongside.

What we have to do is to commence in the top-left corner, and slant down diagonally to the right until we reach the bottom row, multiplying together (that is, ANDing) the three elements encountered along the way. Our first effort produces 0.1.1 (which I haven't dotted, for clarity's sake). Then we move along the top row to the next element, and again we'll slant down to give us "bdc", which we'll add (that is, we'll OR) to what we've already got, so now we've got "0.1.1 + bdc". And once more, along to the top-right element, and slant down to give another block of three literals, viz. "eda". Newcomers will now slide along the top row and into the duplicate matrix, doing exactly the same thing but slanting downwards TO THE LEFT this time, to add yet another three terms to our expression. Once again, I've omitted the dots through the main diagonal, otherwise we wouldn't be able to see anything for dots! To give a complete picture, I'll mention that in normal algebraic operations, the second set would be SUBTRACTED, not ADDED, but everything is ADDED in Boolean algebra.
So now we have

$$\begin{aligned} & 0.1.1 + bdc + eda + 0.d.d + b.a.1 + e.1.c \\ = \; & 0 + bdc + eda + 0 + ab + ce \\ = \; & ab + ce + aed + bcd \end{aligned}$$

and we're not at all surprised to find that this is the same as that found by path-tracing. In actual fact, of course, as we write down the original terms from the determinant, we'd omit completely any which contained a "0", and we'd also omit the "1"s from any term which contained one. This would immediately give us the second of the three expressions above, but without the 0-terms!

That wasn't so bad after all, was it? So now you know about Boolean determinants!

FIFTH-ORDER BOOLEAN MATRICES

Wow! Just look at that heading! Enough to set you running for cover, isn't it? So let's just sneak up on it, and see if we can't lick this one too!

I mentioned that this method gets rougher as the size of the original matrix-a grows, but I think we're brave enough to tackle a 5-by-5. If not, there's always the helicopter back along the trail! All right then, we'll try our hand at the 5 x 5 of Diagram 105a.

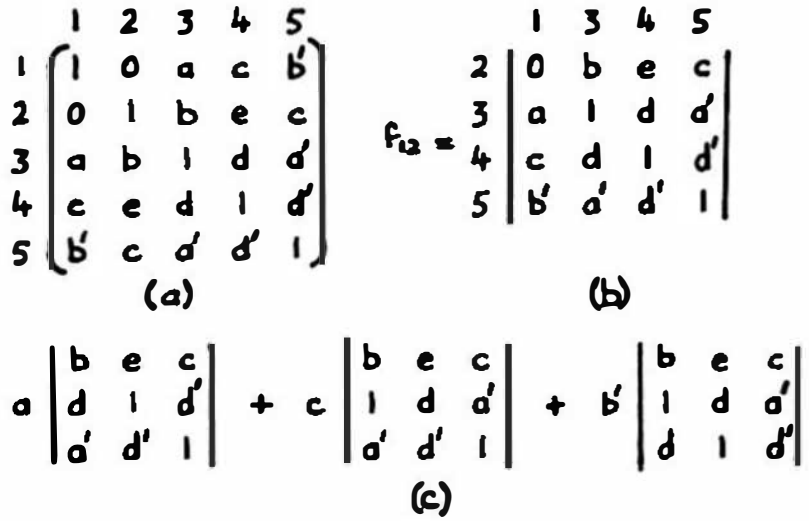


Diagram 105

DERIVING A FOURTH-ORDER DETERMINANT FROM A FIFTH-ORDER MATRIX

First of all, we'll establish that it's f_{12} we're interested in, by eliminating entirely from 105a both row-1 and column-2 to give us the fourth-order determinant of 105b. Observe the convention of indicating a matrix with delimiters to the right and left, with a curved top and bottom, whereas a determinant is delimited with simple straight lines. So ... curved lines mean that the matrix, even a reduced one produced by pivotal condensation, precisely describes its original network. Straight lines indicate that at least one column and row have been COMPLETELY eliminated, and not absorbed into a new matrix, as happens with our first technique. It's possible, with experience, to read fourth-order determinants quite accurately, but as we're essentially lazy at heart we'll reduce ours to the simpler-to-handle third-order variety!

NOW FOR OUR THIRD-ORDER DETERMINANTS

We'll do this scientifically, by selecting ANY SINGLE ROW OR COLUMN for elimination, preferably one with as many 0-entries as possible. The reason for this will become clearer as we proceed! Here we have a choice of either row-2 or column-1 (there is no row-1), so let's select column-1 and loop it in red.

We're now going to work our way down the selected column, one element at a time, and omit the row in which this element occurs. This element will then be multiplied by (ANDed with) the 3-by-3 determinant which remains. In case this all sounds a little confusing, and I've no doubt that it does, just tag along with me as I actually do this. The first element in our selected column is 0, which, as I've just mentioned, has to be multiplied by some determinant. However, as 0 times anything is equal to 0, we've really got nothing to do here. See now why we love to have as many 0s as possible?

Anyway, moving down to the next element, "a", in row-3, we must mentally eliminate row-3 from our 4-by-4 determinant of 105b (keeping in mind that column-1 is already gone, in theory anyway), and create a little 3-by-3, which we've written in 105c, with an "a" to the left, as this is the element involved in the deleted column-1. Then down to the next element in this column, namely "c", eliminate row-4 in our minds, and write THIS remaining 3-by-3 with a "c" to its left, and a "+" sign between the two third-order determinants to indicate that they're going to be ADDED together (that is, ORed). And finally to element b' in column-1 of 105b which gives us the end determinant with a b' to its left. Now re-read my original instructions, and you should be able to follow them this time!

MINOR DETERMINANTS

These final determinants, not being the full determinant of 105b, are individually called "minors". And now comes the tedious task of evaluating these minors, as we did with the 3-by-3 of our earlier example, and multiplying each one by whatever is outside it. This is the time when I wish that the column had more 0s in it! It's got to be done, however, so let's get on with it, first of all setting out the expression in full.

```
  a(b + ed'a' + cd'd + bd'd' + ed + a'c)
+ c(bd + ea'a' + cd' + bd'a' + e + cda')
+ b'(bdd' + ea'd + c + ba' + ed' + cdd)
```

Now comes the horrendous task of multiplying each line of this complex expression by the literal outside each set of parens. But first note that inside each set of parens a single literal occurs, and we can therefore eliminate any term which contains this literal. At the same time, any term which contains the complement of the literal OUTSIDE the parens may be eliminated, because when multiplied together we'll have the situation xx'y, and as we know from our basic Boolean algebra (we do remember, don't we?), xx' = 0, thereby cancelling out this term. We've also got several occurrences of this sort INSIDE the parens! So let's do this reduction first, to give us

```
a(b + ed) + c(bd + cd' + a'bd' + e + a'cd) + b'(a'de + c + d'e + cd)
```

which then multiplies out to

```
ab + ade + bcd + (cd' + a'bcd') + ce + a'cd + a'b'de + (b'c + b'cd) + b'd'e
```

I've highlighted two sets of terms inside parens. where we see that as the smaller term is contained in its entirety within the larger term, the larger term can disappear.

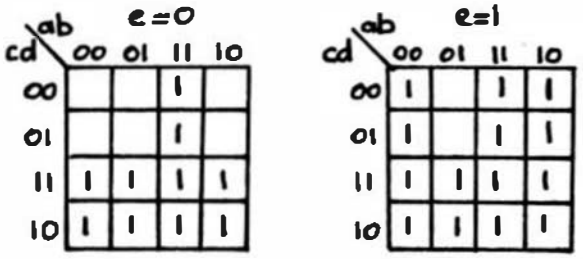


Diagram 106

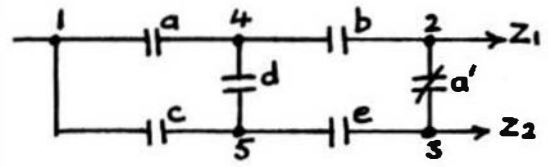
The simplest way to finish off the simplification is to draw up a 5-variable K-map as in Diagram 106, and read out the expression as

$f_{12} = ab + c + b'e$

If this all seems like a lot of work from beginning to end, just imagine what would have been involved if we'd started off with a 6-by-6 matrix. this would have reduced to a 5-by-5 determinant, which would have reduced to five 4-by-4 first minors, EACH OF WHICH would then have reduced to four 3-by-3 second minors, for a grand total of twenty 3-by-3 minors to evaluate and simplify. Can you imagine this? Not that it would be EASY by the Pivotal Condensation method, but I think it would be far, far easier than this!!

MULTIPLE-OUTPUT NETWORKS AND BOOLEAN MATRICES

Now that we're rolling along nicely and smoothly, let's recall my earlier bit of info to the effect that Boolean matrices can keep track of several outputs simultaneously. We'll even complicate things a little, by making our outputs share part of the same network!



(a)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & a & c \\ 0 & 1 & a' & b & 0 \\ 0 & a' & 1 & 0 & e \\ a & b & 0 & 1 & d \\ c & 0 & e & d & 1 \end{bmatrix} \end{array}$$

(b)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & 0 & ce & a + cd \\ 0 & 1 & a' & b \\ ce & a' & 1 & de \\ a + cd & b & de & 1 \end{bmatrix} \end{array}$$

(c)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & b(a + cd) & e(c + ad) \\ b(a + cd) & 1 & a' + bde \\ e(c + ad) & a' + bde & 1 \end{bmatrix} \end{array}$$

(d)

Diagram 107

The circuit of Diagram 107a is a three-terminal network, with power input at node-1, two outputs (Z1 and Z2) at nodes 2 and 3, and two intermediate nodes, 4 and 5. Its Boolean matrix is shown in 107b, so we'll begin a preliminary, partial analysis by eliminating nodes 4 and 5 from matrix-a, leaving only the terminal nodes 1, 2 and 3.

First, let's eliminate node-5 by Pivotal Condensation, to produce matrix-b of 107c, and then node-4 to give matrix-c of 105d. Note that these are MATRICES, not DETERMINANTS, as we haven't actually eliminated anything, but merely CONDENSED our original matrix! In our final third-order matrix-c, element a_{12} details the paths from node-1 to node-2 (but NOT via node-3), a13 the paths *from node-1 to node-3 (but NOT via node-2), and a_{23} the paths from node-2 to node-3 (but NOT via node-1).

EVALUATING INDIVIDUAL PATHS

If we wish to find the expression which describes the complete transmission-function f_{12}, between nodes 1 and 2, we must eliminate node-3 by Pivotal Condensation, to produce

$$\begin{aligned} f_{12} &= b(a + cd) + e(c + ad)(a' + bde) \\ &= ab + bcd + (ce + ade)(a' + bde) \\ &= ab + bcd + a'ce + bcde + abde \\ &= ab + bcd + a'ce \end{aligned}$$

Note that the final term supplies power to Z1 via node-3! In similar vein, in order to find f_{13} we'd eliminate node-2 from 107d instead, and if, for some reason, we were interested in knowing under what conditions nodes 2 and 3 would be connected together, we'd eliminate node-1 instead. The finally cleaned up functions are given below, which you could check for yourselves by actually doing each evaluation.

$$\begin{aligned} f_{13} &= ce + ade + a'bcd \\ f_{23} &= a' + bce + bde \end{aligned}$$

A systematic method of node-elimination such as this gives us the transmission-function between any pair of nodes desired, automatically taking into account ALL possible paths through the network. Our other alternative is to actually trace out the paths in the circuit-diagram, always with the possibility that one or more paths would be overlooked. You'll have noticed that the Laws of Boolean Algebra really come into their own during this reduction process, as the work is made much simpler if the elements are simplified as we go along. For example, when reducing from 107c to 107d, the element c_{13} starts out as

$$ce + (a + cd)(de)$$

and instead of just inserting this complex little beast into element c_{13}, we multiply out and simplify first.

$$= ce + ade + cde \quad \text{(cde vanishes because of ce) to give } e(c + ad)$$

CONNECTION MATRICES AND OUTPUT MATRICES

A matrix such as that of 107b, which contains only one literal per element is known as a "primitive connection matrix", while matrices such as 107c or 107d, which contain more than one literal in any element, are called simply "connection matrices".

There's a third type of matrix, known as an "output matrix", which charts, not the physical connections in the network, but the FUNCTIONS describing the transmissions between nodes. As an example of an output matrix, we could draw up a 3-by-3 for our present circuit, and insert in each element the three transmission-functions just evaluated individually. Such a matrix would, in each element, give the complete transmission-function between the nodes forming its co-ordinates IRRESPECTIVE OF WHETHER SUCH A PATH GOES THROUGH ANOTHER NODE OR NOT, whereas a normal connection matrix describes the connections between these nodes BUT NOT VIA ANY OTHER NODE WHOSE NUMBER APPEARS IN THE MATRIX CO-ORDINATES.

The output matrix for the network of 107a is constructed by inserting the three transmission functions already worked out, as shown next.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & b(a+cd)+d'ce & ce+ade+a'bcd \\ b(a+cd)+a'ce & 1 & d'+be(c+d) \\ ce+ade+d'bcd & d'+be(c+d) & 1 \end{bmatrix} \end{array}$$

Diagram 108

The element d_{12} now gives, not the PATH between nodes 1 and 2, but the CONDITIONS (ALL the conditions) under which transmission will occur between these nodes. Take time out to think this over, so you're reasonably clear about the distinction between the matrices of 107d and 108, even though both of them belong to the SAME network.

That's got the ANALYSIS part of this thing out of the way, so now for SIMPLIFICATION. This wil always be our line of attack, first analysis, then simplification, and then the big one - synthesis.

NETWORK SIMPLIFICATION BY BOOLEAN MATRICES

Are we all ready to move on in our study of Boolean matrices? If so, we'll illustrate the versatility of our new tool by eliminating, by Pivotal Condensation, the non-terminal nodes 3, 4 and 5 from the network of Diagram 109, thus deriving the FUNCTION f_{12} in its simplest form.

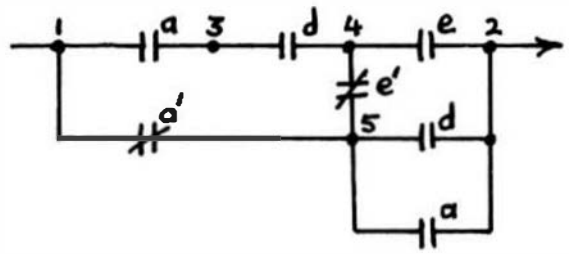


Diagram 109

In our previous examples, no simplification was possible, as was obvious from the final expressions derived. The matrix for circuit 109 and the subsequent step-by-step elimination of its non-terminal nodes are set out in Diagram 110. The finally simplified expression for f_{12} shows that the original network can be completely replaced by a single NO-contact on relay-D, as this is all that's necessary to specify the transmission-function between nodes 1 and 2. Without Boolean matrix techniques, we'd never be able to figure this out from the network, as it's quite a tough job even to figure out its Boolean expression directly!

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 1 & 0 & a & 0 & d' \\ 0 & 1 & 0 & e & a+d \\ a & 0 & 1 & d & 0 \\ 0 & e & d & 1 & d' \\ a' & a+d & 0 & e' & 1 \end{bmatrix} \end{array}$$

(a)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 1 & a'd & a & a'e' \\ a'd & 1 & 0 & e+a+d \\ a & 0 & 1 & d \\ a'e' & e+a+d & d & 1 \end{bmatrix} \end{array}$$

(b)

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & a'd & a+de' \\ a'd & 1 & d \\ a+de' & d & 1 \end{bmatrix} \end{array}$$

(c)

$$\begin{aligned} f_{12} &= a'd + d(a + de') \\ &= a'd + ad + de' \\ &= d(a' + a + e') = d(1 + e') = d \end{aligned}$$

Diagram 110

Suppose, in addition, that there were an additional output being taken off at node-4 to drive some other device, and we desired to know the transmission-function f_{14}, as well as f_{12}. Now we're getting a little complicated, to say the least, but let's go along with this assumption. Node-5 has already been eliminated in 110b, so we'll now proceed to eliminate nodes 2 and 3 (so only 1 and 4 will be left), commencing with node-3 to produce the matrix of Diagram 111a, from which we'll work out the function f_{14}, ending up with "a'e' + d".

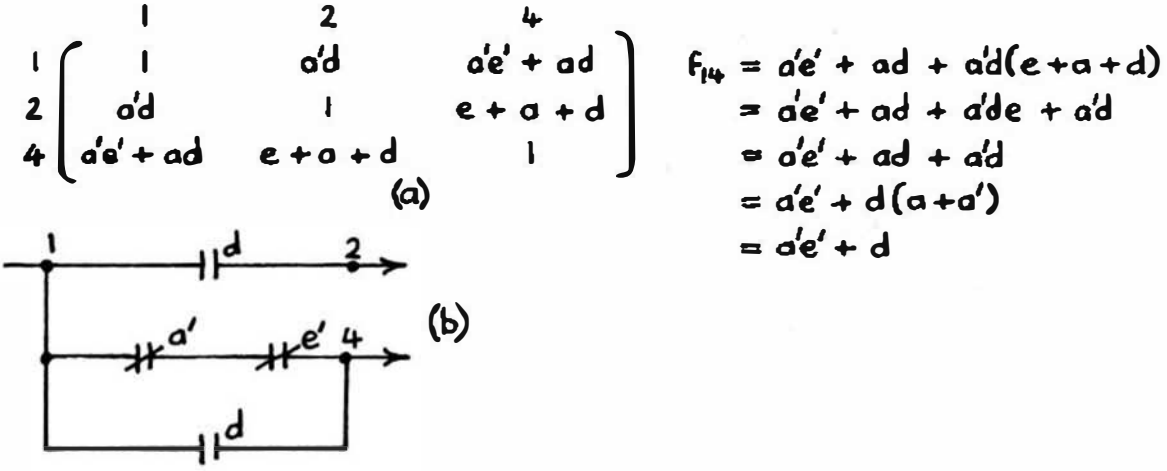


Diagram 111

The COMBINED dual-output circuit of the two simplified functions is shown in 111b. But let's not forget,however, that MATRIX CONTRACTION YIELDS THE SIMPLEST FUNCTION, AND NOT NECESSARILY THE SIMPLEST CIRCUIT. If you don't believe me, just go back and look at the FUNCTION derived from matrix contraction in diagram 104, and the circuit diagram which we could reasonably expect to construct from it. The simplest CIRCUIT is, of course, the original one of 101, but the simplest FUNCTION describing it is that which describes the equivalent circuit of Diagram 100b. All clear?

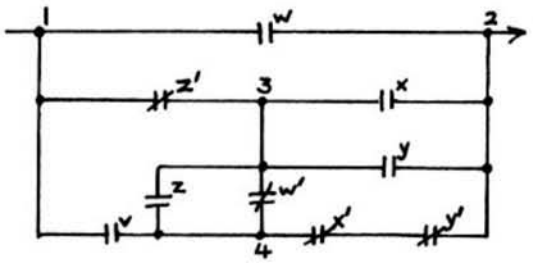
MORE CHIT-CHAT

While matrix contraction yields the simplest transmission-function, matrix EXPANSION, which we'll be studying shortly, gives us a simplified circuit-diagram, if one exists. Don't be confused by the fact that matrix contraction has just given us a MUCH SIMPLER network, as it often happens that the simplest function is co-incidentally the simplest circuit also. If this DOES occur, remember that it's just an accidental side-benefit derived from the true purpose of contraction, and you'll never be really sure whether it's truly the simplest circuit. Unless, of course, you end up with just a few simple contacts, and you can tell at a glance that it can't be reduced further.

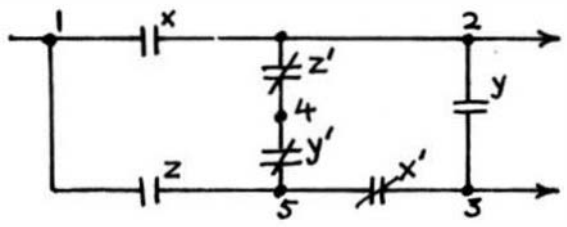
And now it's TEST-time again for all you lucky people, so let's have a go at the following, but think carefully before you answer the question posed in Problem 2!

TEST FIFTEEN

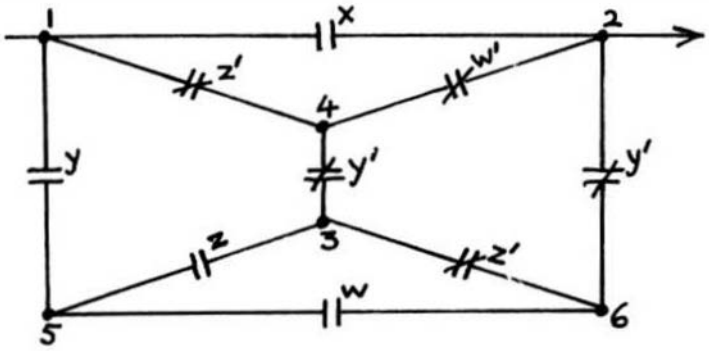
1. Evaluate f_{12} by means of Laplace's Development, and draw the final circuit.



2. Nodes 1, 2 and 3 are terminal nodes. Eliminate nodes 4 and 5 by Pivotal Condensation, and draw the final circuit diagram. Is this a simpler circuit?



3. Evaluate f_{12} by means of Pivotal Condensation and draw the final circuit.



CAMP TIME AGAIN

Come on now, and admit that although this has been a VERY complicated few miles, this new stuff is nowhere near as terrifying as our encounter with the Laws-of-Boolean Swamp. And just think what it can do for your design skills!

Anyway, while you're working at these three problems, I'm debating whether to lead you into another of those word-logic problems, similar to that involved in Uncle Fred's experiences with I-asku of the M'bulyan tribe. I'll let you know when we're ready to move off once more.

... End of Mile 20, and all gathered round marker Mile-21.

+++

The Macintosh Section

Reserved as

A place for your thoughts

And ours.....

Mac-Watch

This and That

Reviews of MaxPage and QuickLetter

By James E. Law
1806 Rock Bluff Rd.
Hixson TN 37343

This month I want to briefly review several new software packages starting with MaxPage which its distributor, Applied System and Technologies, describes as "the affordable page makeup program."

A Review of MaxPage

A logical part of us knows that you don't get something for nothing but in spite of this, we search for that rare deal where we get a treasure for pennies. Occasionally it works out but most of the time you get what you pay for. When I received a piece of software which advertised page layout capabilities for a suggested retail of $89, my curiosity was peaked. Could it really deliver? Let's find out.

When you first open MaxPage, a blank page is presented with a set of menu bars. The page may be in landscape or portrait orientation. You may activate a grid (maximum size of 17" by 17") for ease in locating text or other items on the page. Oddly, the default settings are for the grid to print out when you print your page, but it may be ordered not to print. It is specially marked to facilitate the layout of bifolds and trifolds. There is only one view of the page - full size - and no page preview is provided. This is an unforgivable omunission for a page layout program!

Working With Text

To work with text, you first create a text block by clicking and dragging the cursor. Each text block has a drag handle for use in re sizing the block. You can move text blocks around and position them anywhere you like on a page. There can be up to 75 text blocks per page.

Upon sizing and positioning a text block, you can enter and edit text in the usual fashion. Multiple fonts and sizes within a text block are supported. Text can be typed into text blocks or imported from other applications. Text can be aligned left, right, or center but may not be flush justified. No tabs are available. MaxPage allows considerable control over line spacing. You can adjust the leading above or below the selected line or have leading automatically adjusted based on a number of options.

If you enter or import more text than a text block can hold, it stacks below the visible area of the text block. The hidden portion of the text can be seen by re sizing the block. This excess portion of the text can also be made to wrap over to a different text block on the same or a different page. This is a manual process that might work okay for a simple one or two page document but would not be practical for a more elaborate project.

Blocks are also used to accept and position PICT (draw-type) or PNTG (bit mapped) images. The image is automatically scaled to fill the active picture block. If you re size the block, the image is re sized. MaxPage takes a highly unusual approach to handling images in that the image is referenced to its location on your drive and is *not* really imported into MaxPage. Every time the page

is re drawn, the image is re-imported. This means that the image file must stay on the same disks as the MaxPage file. For a complex image, a noticeable delay occurs in redrawing the page. To avoid this delay, you may elect an option in which images are neither displayed or printed.

Backgrounds

MaxPage allows you to import a full page PICT image into your document as a "background." Unlike the image handling covered above, backgrounds are actually imported into and become part of your MaxPage document. This feature is intended to contain master elements (e.g., borders, letter heads, or logos) or to serve as a design templet.

Conclusion

After my look at MaxPage, I conclude that you get what you pay for. . . maybe. What MaxPage has going for it is its cheap price. Disadvantages include the lack of page preview, no text tabs or justification, and nonstandard ways of handling graphics. If all you need is to lay out simple, occasion. single page, perhaps MaxPage will serve your needs. All in all, however, I cannot recommend this product..

A Review of QuickLetter

Over recent years there have been a number of mini word processors in desk accessory form for the MacIntosh. The quality and number of features in those programs varied widely but with a few being quite powerful. I have just received QuickLetter from Working Software, Inc., and it is a jewel.

What it Does

QuickLetter is primarily designed for one thing -- writing letters. To that end it provides for setting up standard stationary (with letter heads and logos), maintaining an address book, importing addresses into your correspondence, and printing letters and envelopes.

When you open QuickLetter, you may use the default "stationary" (to be discussed later), open other McWrite or QuickLetter documents or create a new document. Your document then appears as does a ruler (which may be hidden) with various tools and a set of menu bars. Your document may be either landscape or portrait orientation in any standard size.

QuickLetter's text tools are fairly standard so I won't detail them. A feature that I like is the ability to use text size up to 200. Also, in QuickLetter, you have more control over setting margins and line spacing than in MacWrite.

QuickLetter allows you to set up standard stationary to speed up the letter writing process. Such stationary already contains standard graphics features (e.g., letter heads borders and/or logos) margin and tab settings, and text specifications. One such stationary design may be designated as the default stationary and will automatically appear when you start QuickLetter.

You may import graphics for use as a logo or letterhead. Whatever you import is automatically placed in the upper left hand corner of the page and cannot be moved. This lack of flexibility is a significant weakness in QuickLetter. Considerable trial and error may be necessary to design the graphic so that when pasted into QuickLetter, it is corrected positioned.

QuickLetter contains a handy address book for in putting addresses to your letter or envelope. This feature also can be used to print labels. You may search the address book for key words such as a part of a name.

The print preview feature for QuickLetter is especially nice. Not only can you easily preview your letter and envelope as they will print but you can move the text around on the page independent of the letterhead to visually position it as you like.

The Bottom Line

In summary, this is an unexpected nice product. I keep finding useful features that I didn't expect to find. If the imported graphics could be more easily located, it would be perfect. But that weakness can be worked around and so, on the balance, QuickLetter's strengths outweigh its weaknesses and is recommended.

EOF

PROGRAMMING LANGUAGES

PL/9 from Windrush Micro Systems -- By Graham Trott. A combination Editor Compiler Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide.
FLEX, SK-DOS, CCF - $198.00

PASC from S.E. Media - A FLEX9, SK-DOS Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package comes complete with source (written in PASC) and documentation.
FLEX, SK-DOS $95.00

WHIMSICAL from S.E. MEDIA Now supports *Real Numbers*. "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. *Normally produces 10% less code than PL/9.*
FLEX, SK-DOS and CCF - $195.00

KANSAS CITY BASIC from S.E. Media - *Basic for Color Computer OS-9* with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFT$, RIGHT$, MID$, STRING$, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.
CoCo OS-9 $39.95

C Compiler from Windrush Micro Systems by James McCosh. Full C for FLEX, SK-DOS except bit-fields, including an Assembler. *Requires the TSC Relocating Assembler if user desires to implement his own Libraries.*
FLEX, SK-DOS, CCF - $295.00

C Compiler from Introl -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
FLEX, SK-DOS, CCF, OS-9 (Level II ONLY), UniFLEX - $575.00

PASCAL Compiler from Lucidata -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
FLEX, SK-DOS and CCF - $190.00

OmegaSoft PASCAL from Certified Software -- Extended Pascal for systems and real-time programming.
Native 68000/68020 Compiler, $575 for base package, options available. For OS-9/68000 and PDOS host system.
6809 Cross Compiler (OS-9/68000 host) $700 for complete package.

KBASIC - from S.E. MEDIA -- A "Native Code" BASIC Compiler which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, SK-DOS, CCF, OS-9 Compiler /Assembler $99.00

CRUNCH COBOL from S.E. MEDIA -- Supports large subset of ANSII Level 1 *COBOL* with many of the useful Level 2 features. Full FLEX, SK-DOS File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System. *A very popular product.*
FLEX, SK-DOS, CCF - $99.95

FORTH from Stearns Electronics -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer ONLY - $58.95

FORTHBUILDER is a stand-alone target compiler (crosscompiler) for producing custom Forth systems and application programs.
All of the 83-standard defining words and control structures are recognized by FORTHBUILDER.
FORTHBUILDER is designed to behave as much as possible like a resident Forth interpreter/compiler, so that most of the established techniques for writing Forth code can be used without change.
Like compilers for other languages, FORTHBUILDER can operate in "batch mode".
The compiler recognizes and emulates target names defined by CONSTANT or VARIABLE and is readily extended with "compile-time" definitions to emulate specific target words.
FORTHBUILDER is supplied as an executable command file configured for a specific host system and target processor. Object code produced from the accompanying model source code is royalty-free to licensed users.
FLEX, CCF, SK-DOS - $99.95

EDITORS & WORD PROCESSING

JUST from S.E. Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.
* Now supplied as a two disk set:
Disk #1: JUST2.CMD object file,
JUST2.TXT PL9 source:FLEX, SK-DOS - CCF
Disk #2: JUSTSC object and source in C:
FLEX, SK-DOS, OS-9, CCF
The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C

source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p ,u ,y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!

Disk (1) - PL9 FLEX only. FLEX, SK-DOS & CCF - $49.95

Disk Set (2) - FLEX, SK-DOS & CCF & OS-9 (C version) - $69.95

OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.

Regular FLEX, SK-DOS $129.50

** SPECIAL INTRODUCTION OFFER * $79.95*

SPECIAL PAT/JUST COMBO (with source)

FLEX, SK-DOS $99.95

OS-9 68K Version $229.00

SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.

FLEX, SK-DOS $69.95

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entire lines, etc. Complete with over 25 different CRT terminal configuration overlays.

FLEX, CCF, SK-DOS $39.95

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX, SK-DOS or SSB-DOS, OS-9 - $175.00

SPELLB "Computer Dictionary" from S.E. Media -- **OVER 150,000 words!** *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX, SK-DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

FLEX, SK-DOS and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

NEW PRICES 6809 CCF and CCO - $99.95,

FLEX, SK-DOS or OS-9 - $179.95, UniFLEX - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.

NEW PRICES 6809 CCF and CCO - $69.95,

FLEX, SK-DOS or OS-9 - $99.95, UniFLEX - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.

NEW PRICES 6809 CCF and CCO - $59.95,

FLEX, SK-DOS or OS-9 - $79.95, UniFLEX - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!

FLEX, SK-DOS or OS-9 - $329.95, UniFLEX - $549.95

OS-9 68000 $695.00

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems

FOR 6809 FLEX or SK-DOS (5/8")

Up to 32 groups/fields per record! Up to 12 character file names! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

FLEX 6809 DIAGNOSTICS Disk Drive Test ROM Test, RAM Test

INTRODUCTION

The following utilites are available on 68 Micro Journal *Reader Service Disk #34*. However, they are being published here as well because of the sharing spirit of the author:

Emery Korpi
40 Beatrice Ave.
Syosset, NY, 11791.

I, and thousands of loyal readers thank you Emery for your willingness to share your efforts with all of us. It is that type of spirit that has allowed us to survive. As a contributor supported magazine we urge all our readers to do likewise. If so, we will be around a long time to come.
Again, thanks - Emery!

DMW

DISC DRIVE TEST (DDT)
NOTES:

INTRODUCTION -

THIS PROGRAM HAS BEEN WRITTEN TO AID IN THE ALIGNMENT OF 5 INCH* DISK DRIVES UNDER THE FLEX OP-SYSTEM. IT ALLOWS FOR TESTING OF DRIVES 0 THROUGH 3 WITH THE WD179X SERIES OF DISK CONTROLLERS IN ANY OF THE SYSTEM I/O SLOTS. THE PROGRAM MAY BE USED WITH EITHER 1MHZ OR 2MHZ CLOCKED SYSTEMS AND IS COMPATIBLE WITH DC-1 TO DC-4 OR FD-2 CONTROLLERS THIS PROGRAM HAS THE FOLLOWING TEST CAPABILITIES.

1. DRIVE READY SIGNAL RESPONSE TIME
2. HEAD LOAD DELAY TIME
3. BUSY SIGNAL RESPONSE TIME
4. DRIVE ROTATIONAL SPEED TEST
5. INDEX PULSE WIDTH TEST
6. TRACK 0 SWITCH ADJUSTMENT
7. DRIVE STEP TIME
8. TEST OF ALL DISK SECTORS
9. RADIAL ALIGNMENT
10. READ/WRITING/MODIFYING DATA ON ANY SECTOR.

THIS PROGRAM ALSO CAN OPERATE WITH THE DYSAN DIGITAL DIAGNOSTIC DISKS IN ORDER TO FIND;

1. RADIAL TRACK ALIGNMENT ERRORS (IN MILLI-INCHES)
2. STEPPER MOTOR ACCURACY AND REPEATABILITY PROBLEMS
3. REDUCED AMPLITUDE SIGNAL PROBLEMS
4. AZIMUTHAL HEAD ALIGNMENT ERRORS

A. STARTING THE PROGRAM -

THE PROGRAM MAY BE STARTED IN EITHER OF THREE WAYS. THESE ARE;

1. DDT - THE OPERATOR WILL BE PROMPTED FOR THE DRIVE NUMBER (0-3), THE SYSTEM CLOCK SPEED (1 OR 2 MHZ) AND THE I/O QLOT THAT THE CONTROLLER IS INSTALLED IN.

2. DDT,# - THE PROGRAM WILL TEST DRIVE NO '#' AND ASSUMES A 1 MHZ CLOCK RATE AND I/O SLOT ONE. THESE DEFAULT VALUES MAY BE CHANGED IN THE SOURCE PROGRAM WHICH IS SUPPLIED.

3. DDT,>file - THE PROGRAM WILL BE PROMPTED THE SAME AS IN DDT AND A COPY OF ALL TERMINAL DATA WILL BE WRITTEN INTo TEXT FILE file.

B. TEST OPTIONS -

THE OPERATOR OF THIS PROGRAM HAS THE OPTION OF HALTING AFTER EACH TEST. IF THIS OPTION IS NOT TAKEN, THE PROGRAM WILL PROCEED AUTOMATICALLY UNTIL A FAILURE OR KEYBOARD ENTRY IS ENCOUNTERED; AT WHICH POINT IT WILL HALT. THE PROGRAM WILL THEN GIVE THE OPERATOR THE FOLLOWING OPTIONS:

1. RETRY - ENTERING '1' WILL CAUSE THE PROGRAM TO RE-EXICUTE THE CURRENT TEST

2. CONTINUE - ENTERING '2' WILL CAUSE THE PROGRAM TO CONTINUE TO THE NEXT TEST

3. EXIT - ENTERING '3' WILL CAUSE A RETURN TO THE OP-SYSTEM

C. DISK DRIVE TESTING PRINTOUT FORMAT

I. DRIVE MOTOR STATUS(E0X8)
   A.READY RESPONSE TIME = XXXX.XX MSEC
   B.BUSY RESPONSE TIME = XX USEC

C.HEAD LOAD DELAY TIME = XXXX.XX MSEC
D.WRITE PROTECT = Y/N
E.INDEX PULSE COUNT = XX S/B GT 2

THESE TESTS ARE PERFORMED BY READING THE DISK CONTROLLER STATUS WHICH IS FOUND AT MEMORY LOCATION E0X8.
- READY RESPONSE TIME IS THE TIME THAT THE READY SIGNAL TAKES TO GO HI. IT IS THE TIME REQUIRED FOR THE DRIVE MOTOR TO REACH ITS DESIGNATED SPEED.
- BUSY RESPONSE TIME IS THE TIME NECESSARY FOR THE CONTROLLER TO DIGEST AN INSTRUCTION AND INDICATE IT IS BUSY.
- HEAD LOAD DELAY TIME IS THE TIME ALLOTTED FOR THE HEAD TO LOAD BEFORE READING OR WRITING CAN PROCEED.
- WRITE PROTECT INDICATES IF THE DISK IS WRITE PROTECTED.
- INDEX PULSE COUNT COUNTS THE NUMBER OF INDEX PULSES WHICH OCCUR IN THE ALLOTTED TIME SPAN. FAILURE INDICATES NO DISK IN DRIVE, MOTOR FAILURE OR CLUTCH FAILURE.

II. DRIVE SPEED TEST
PERIOD = XXXX.XX MSEC S/B 200 +/- 1 MSEC

THIS TEST MEASURES THE ROTATIONAL SPEED OF THE DISK BY MEASURING THE TIME PERIOD OF THE INDEX PULSE. MISADJUSTMENT IS A COMMON CAUSE OF SOFT DISK ERRORS. THE SPEED MAY BE ADJUSTED AND CHECKED WITH THE 'RETRY' OPTION.

III.INDEX PULSE WIDTH
WIDTH = XXXX.XX MSEC S/B 2.5-5.5 MSEC

THIS IS THE PULSE WIDTH OF THE SIGNAL GENERATED WHEN THE INDEX HOLE PASSES IN FRONT OF THE INDEX DETECTOR CIRCUITRY OF THE DRIVE.

IV.TRACK 0 SWITCH TEST
A. STEP TIME FROM TRACK 0 TO SWITCH = XXXX.XX
B. STEP TIME FROM TRACK 1 TO SWITCH = XXXX.XX
C. SWITCHPOINT(A/A+B) = XX% S/B 5% TO 95%

THIS TEST VERIFIES THE POSITION OF THE TRACK 0 MICROSWITCH IS CORRECTLY ADJUSTED. THE 'TRACK 0' SIGNAL STOPS THE CONTROLLER FROM STEPPING WHEN A RESTORE TO TRACK 0 COMMAND IS ISSUED.

THIS TEST DETERMINES THE TRACK 0 SWITCHPOINT POSITION BY TIMING MEASUREMENTS. TIME MEASUREMENTS ARE MADE STARTING FROM THE STEP COMMAND UNTIL THE SWITCH TOGGLES WHEN:

A. STEPPING FROM TRACK 0 TO 1 AND
B. STEPPING FROM TRACK 1 TO 0.

THE SWITCHPOINT IS THE RATIO OF ONE MEASUREMENT TO THE SUM OF THE MEASUREMENTS (IE A/A+B).

FOR DRIVES USING PHOTO DIODE POSITION DETECTOR SWITCHES, THIS TIMING TEST METHOD ACCURATELY REFLECTS THE PHYSICAL SWITCHPOINT POSITION. MEASUREMENTS OF 40-60% ARE TYPICAL FOR THIS TYPE OF DRIVE.

FOR DRIVES WITH A MECHANICAL SWITCH, THE MEASUREMENT IS INFLUENCED BY SWITCH SPRING TENSION. TYPICAL MEASUREMENTS OF 5-20% MAY BE OBTAINED WITH A PHYSICAL SWITCHPOINT MIDWAY BETWEEN TRACKS 0 AND 1.

V. STEP RATE TEST
TRACK STEP TIME = XX.XX MSEC S/B LT 40 MSEC

THIS TEST FINDS THE DRIVE STEP TIME. THIS IS THE SMALLEST TIME NECESSARY FOR THE DRIVE TO STEP FROM ONE TRACK TO THE NEXT.

THIS TEST IS PERFORMED BY STEPPING BETWEEN TRACKS 32 AND 0 AT DIFFERENT RATES. THE RATE IS CONSIDERED TO BE SUCCESSFUL IF THE TRACK 0 SWITCH HAS BEEN SET. THE STEP RATE IS FOUND BY A NINETH ORDER BINARY SEARCH AS FOLLOWS:

```
TIME:=DELTA:=20.48MSEC
FOR I = 1 THRU 9
  DELTA := DELTA/2
  IF SUCCESSFUL THEN
    TIME := TIME - DELTA
  ELSE
    TIME := TIME + DELTA
  END IF
END FOR
```

THIS TECHNIQUE YIELDS A MEASUREMENT RANGE OF 0 TO 41 MSEC WITH AN ACCURACY OF 0.08 MSEC.

VI.DOUBLE DENSITY STATUS(E0X4)
A. RESTORE USING INTRQ(D6) GO/NG
B. READ SECTOR USING DRQ(D7) GO/NG
C. NO. BYTES READ = XXXX S/B 256 BYTES

SOME SYSTEMS USE THE STATUS WORD AT MEMORY LOCATION E0X4 FOR DISK READ AND WRITE OPERATIONS. THIS TEST CHECKS THE FUNCTIONALITY OF THE TWO SIGNALS ASSOCIATED WITH THIS STATUS WORD.

VII. RADIAL ALIGNMENT TEST
1. CHECK ALL TRACKS
2. ALIGN
3. EXIT
4. CHECK SINGLE TRACK
5. NEXT TEST
6. PRINT SYMBOL TABLE

- CHECK ALL TRACKS OR A SINGLE TRACK PRINTS, FOR EACH SECTOR, AN '*' FOR 256 BYTE SECTORS OR '.' FOR 128 BYTE SECTORS. THE ERROR CODE IS PRINTED FOR A SECTOR WITH AN ERROR. THE ERROR CODES MAY BE PRINTED BY ENTERING OPTION '6'. THE TRACKS ARE TESTED OUT OF ORDER, TO TEST FOR STEPPER MOTOR SKIP AND DIRECTIONAL OFFSETS.
- SECTOR SYMBOL TABLE:
* 256 BYTE SECTOR OK
. 128 BYTE SECTOR OK
U 0-127 BYTES READ INTO BUFFER, STATUS OK
B 129-255 BYTES READ INTO BUFFER, STATUS OK
O OVER 256 BYTES READ INTO BUFFER, STATUS OK
T TRACK NO. ERROR, ON WRONG TRACK
N NO DATA ON TRACK OR UNFORMATTED TRACK
S MISSING SECTOR
C ID FIELD CRC ERROR
A ADDRESS MARK ERROR IN DATA FIELD
D DATA FIELD CRC ERROR

- ALIGNMENT MAY BE PERFORMED WITH EITHER: AN OP-SYSTEM DISK (OR DISK FROM ALIGNED DRIVE) OR WITH A DYSAN DIGITAL DIAGNOSTIC DISK.
- USING THE OP-SYSTEM DISK:
1 ADJUST THE STEPPER MOTOR CLOCK WISE UNTIL ABOUT HALF OF THE SECTORS HAVE ERRORS AND MARK THIS POSITION.
2. ADJUST THE STEPPER MOTOR COUNTER CLOCKWISE UNTIL HALF OF THE SECTORS HAVE ERRORS AND MARK THIS POSITION.
3. CENTER THE STEPPER MOTOR BETWEEN THE TWO MARKED POSITIONS.
- USING THE DIGITAL DIAGNOSTIC DISKETTE(DDD)

A. REDUCTION OF SIGNAL STRENGTH TEST
1. 5/12 AMPLITUDE - ALTERNATE SECTORS OFF SET BY 7 MILS
2. 4/12 AMPLITUDE - ALTERNATE SECTORS OFF SET BY 8 MILS
3. 3/12 AMPLITUDE - ALTERNATE SECTORS OFF SET BY 9 MILS

THE HEAD PICKS UP A SIGNAL IN A 12 MIL WIDE TRACK ON THE DISK. ALTERNATE SECTORS ARE DISPLACED BY 7, 8 AND 9 MILS ON TRACKS 21, 24 AND 27 RESPECTIVELY TO PRODUCE AN EFFECTIVE REDUCTION IN SIGNAL STRENGTH. THIS TEST CHECKS FOR WEAK DRIVE ELECTRONICS, HUB CENTERING, STEPPER PLAY, NEED FOR HEAD CLEANING AND MISADJUSTED HEAD LOAD PADS.

B. TRACK OFFSET IN MILLI-INCHES

TRY	STEP DIRN=IN	STEP DIRN=OUT	LIMITS	GO/NG
1.	+XX.X MILS	+XX.X MILS	+/-1.5	GO
2.	+XX.X MILS	+XX.XMILS	+/-1.5	GO
3.	+XX.X MILS	+XX.X MILS	+/-1.5	GO
4.	+XX.X MILS	+XX.X MILS	+/-1.5	GO
5.	+XX.X MILS	+XX.X MILS	+/-1.5	GO

THERE ARE SIX TRACKS AVAILABLE TO TEST RADIAL ALIGNMENT. THESE ARE TRACKS 0, 5, 16, 19, 30 AND 39. THIS TEST CHECKS THE SELECTED TRACK WITH PROGRESSIVE OFFSETS AND CALCULATES THE DISTANCE THE HEAD IS OFF OF CENTER. THIS IS DONE FROM TWO DIRECTIONS IN ORDER TO CHECK FOR HYSTERISIS OR STEP DIRECTION DEPENDENT PLAY. THE MEASUREMENT RANGE IS -6 TO +6 MILS WITH AN ACCURACY OF +/-0.5 MILS.

C. AZIMUTHAL HEAD ANGLE TEST
HEAD ANGLE = XX.X MIN S/B 0 +/- 12 MIN

THE DISK HEAD READS AND WRITES ON THE RADIAL CENTERLINE OF THE TRACK. THIS TEST MEASURES THE SKEW ANGLE OF THE HEAD RELATIVE TO THE TRACK CENTERLINE BY TESTING SECTORS WITH PROGRESSIVE ANGLE OFFSETS ON TRACK NO. 34. MEASUREMENT RANGE IS -18 TO +18 MINUTES WITH AN ACCURACY OF +/-1.5 MINUTES.

VIII. READ/WRITE SECTOR TEST
1. PATTERN 1
2. PATTERN 2
3. MODIFY BUFFER(M)
4. EXIMINE BUFFER(E)
5. READ SECTOR(R)
6. WRITE SECTOR(W)
7. NEXT SECTOR(N)
8. RETURN TO FLEX(F)
9. HELP(H)

- PATTERN 1 INITIALIZES THE BUFFER TO HEXADECIMAL BDBDBD...
- PATTERN 2 INITIALIZES THE BUFFER TO HEXIDECIMAL 242424...
- MODIFY BUFFER ALLOWS EDITING OF THE SECTOR BUFFER USING:
N = NEXTBYTE
L = LAST BYTE
RTN = EXIT MODIFY
- EXIMINE BUFFER PRINTS THE 256 BYTE SECTOR BUFFER IN A HEXIDECIMAL AND ASCII FORMAT
- READ SECTOR READS THE SELECTED SECTOR INTO THE SECTOR BUFFER AND DISPLAYS THE SECTOR BUFFER
- WRITE SECTOR WRITES THE CURRENT CONTENTS OF THE SECTOR BUFFER INTO THE SELECTED SECTOR
- NEXT SECTOR READS THE NEXT SECTOR IN THE FILE CHAIN
- RETURN TO FLEX ENDS THE PROGRAM
- HELP DISPLAYS ALL TEST VIII TEST OPTIONS

*NOTE-SOURCE PROGRAM MODIFICATIONS ARE NECESSARY TO TEST 8 INCH DISK DRIVES.

'DIGITAL DIAGNOSTIC DISKETTE' IS A TRADEMARK AND 'DYSAN' IS A REGISTERED TRADEMARK OF DYSAN CORP, 1244 REAMWOOD AVE, SUNNYVALE, CA, 94086 : 408-734-1634; 800-551-9000

DYSAN DIGITAL DIAGNOSTIC DISKS PART NUMBERS:
5.25" 48TPI SSSD(#508-100) SSDD(#508-200) DSSD(#508-300) DSDD(#508-400)
5.25" 96TPI SSSD(#506-100) SSDD(#506-200) DSSD(#506-300) DSDD(#506-400)
8.00" 48TPI SSSD(#808-100) SSDD(#808-200) DSSD(#808-300) DSDD(#808-400)

'FLEX' IS A TRADEMARK OF TECHNICAL SYSTEMS CONSULTANTS

EOF

DDT

```
*********************************************
* DISK DRIVE AND CONTROLLER TEST PROGRAM    *
* (5.25 DRIVES, DC-1 TO DC-4 CONTROLLERS)   *
*                                           *
* THIS PROGRAM TESTS SINGLE OR DOUBLE DENS  *
* DISKS OPERATE ON THE SPECIFIED DRIVE.     *
* THIS PROGRAM ALLOWS FOR TRK 0 ADJUSTMENT, *
* ROTATIONAL SPEED ADJUSTMENT AND RADIAL    *
* ALIGNMENT USING A FORMATTED DISK FROM AN  *
* ALIGNED DRIVE OR A DIGITAL DIAGNOSTIC DSK *
*                                           *
* EXAMPLE:                                  *
*   DDT,0        (TESTS DRIVE 0)            *
*   DDT          (ALLOWS CHOICE OF          *
*                 CONTROLLER I/O SLOT)      *
*   DDT > FILE   (BUILDS PRINT FILE)        *
*                                           *
* WRITTEN BY: EMERY KORPI                   *
*             40 BEATRICE AVE               *
*             SYOSSET,NY 11791              *
*                                           *
* VERSION-2 ; DATE-8/87                     *
*********************************************

*        REFRENCES
*6502 ASSEMBLY LANGUAGE SUBROU-
TINES
*BY LANCE A. LEVENTHAL AND WINTHROP SAVILLE
*PUB. OSBORNE/MCGRAW-HILL 1982

*THE FLEX DISK OPERATING SYSTEM
*BY TECHNICAL SYSTEMS CONSULTANTS, INC

*DR. DOBB'S JOURNAL
*PEOPLE'S COMPUTER COMPANY
*VOLUME 8, ISSUE 12, DECEMBER 1983

*'68' MICRO JOURNAL
*COMPUTER PUBLISHING INC.
*2/84, 6/84, 6/84 ISSUES

*NOTE: TO CUSTOMIZE THE DEFAULT MODE FOR
*YOUR SYSTEM, CHANGE THE SYSTEM
*DEPENDENT VARIABLES.

*        ******************
*        * FLEX LOCATIONS *
*        ******************

WARM$  EQU  $CD03
SBUG   EQU  $F814
FMS    EQU  $D406
FMSCLS EQU  $D403
FCB    EQU  $C840
RPTERR EQU  $CD3F
GETFIL EQU  $CD2D
SETEXT EQU  $CD33
TTYEOL EQU  $CC03
NXTCH  EQU  $CD27
LNPNTR EQU  $CC14
MEMEND EQU  $CC2B
PSTR   EQU  $CD1E
GETCHR EQU  $CD15

DRVTST ORG  $1000
       BRA  START0
VN     FCB  2        VERSION
START0 LBRA START

*        **********************
*        * SYS DEPEND VARS.  *
*        **********************

TRMSTA EQU  $E004    TERMINAL STATUS
TRMDAT EQU  $E005    TERMINAL DATA
DRVADD FDB  $E010    DISK CONTROL ADDR
CLKFLG FCB  0        0=1MHZ,1=2MHZ
T7SECT FCC  $0A,$14,$12,$24
*            SSSD,DSSD,SSDD,DSDD SECTORS

*        **********************
*        * PROGRAM VARIABLES *
*        **********************

PADDR  FDB  $3000
PRTFLG FCB  0
SAVLNP RMB  1
DRVNO  FCB  1
DRVCMD FCB  1
CNTR   FCB  1
DIGNO  RMB  3
COUNT  RMB  3
TEMP   FCB  1
TEMP2  FCB  1
TEMP3  FCB  1
TIME   RMB  2
BLSB   RMB  2   BINARY LSB
DMSB   RMB  2   DECIMAL MSB
BNCTIM FCB  1   BOUNCE TIME
ERRFLG FCB  1   ERROR FLAG
RETRY  RMB  2   RETEST ADDR
DVR    RMB  3   DIVISOR
DVD    RMB  3   DIVIDEND
T6FLG  RMB  1   TEST 6 PASSED
DDFLG  RMB  1   TEST 6 GO FLG
MAXSEC RMB  1   # SECS/TRK
FORMAT RMB  1   DISK FORMAT
T7OPTN RMB  1   T7 OPTION
DNSFLG RMB  1   1=DD
TRK    RMB  1   TRACK NO
SEC    RMB  1   SECTOR NO
ADDR   RMB  2   MEMORY ADDRESS
LIMT   RMB  2   LO LIMIT
STPSIZ RMB  2   CONTROLLER STEP SIZE
STPREF RMB  2   TIME TO TRK0 SWITCH
STPCMP RMB  2   T5 MEASUREMENT
RSLTN  RMB  1   OFFSET RESOLUTION
T7CNT  RMB  1   LOOP COUNTER
PCNT   RMB  1   POSITIVE OFFSET
NCNT   RMB  1   NEGATIVE OFFSET
LENGTH RMB  1   BYTES TO DIV
LCOUNT RMB  2
HIDE1  RMB  3
HIDE2  RMB  3
RMDPTR RMB  2   REMAINDER ADDR
TMPPTR RMB  2   TEMP ADDR
DVRPTR RMB  2   DVR POINTER
DVDPTR RMB  2   DVR POINTER

*        **********************
*        * PROGRAM MESSAGES *
*        **********************

TITLE  FCC 'DISK DRIVE TEST PROGRAM [REV-8/87]',4
USGERR FCC 'Usage: DDT(,drive)( > printfile)',4
SLTMSG FCC 'DISK CONTROLLER IN SLOT (0-7,1=STD)? ',4
SLTERR FCC 'INVALID DISK ADDRESS, RETRY(Y/N)? ',4
DRVMSG FCC 'DRIVE NO.(0-3)? ',4
CLKMSG FCC 'SYSTEM CLOCK RATE(1-1MHZ,2-2MHZ)? ',4
HLTMSG FCC 'HALT AFTER EACH TEST (Y/N)? ',4,
DSKMS1 FCC 'INSERT DISK IN DRIVE '
DSKMSH FCC ' , HIT RTN TO CONTINUE',4
```

```
T1MSG  FCC $0D,$0A,'I. DRIVE MOTOR STATUS (E0X8)',4
T1AMSG FCC '   A. READY RESPONSE TIME = '
T1AMSH FCC '   0000.00 MSEC S/B LT 620 MSEC',4
T1BMSG FCC '   B. BUSY RESPONSE TIME = '
T1BMSH FCC '   0000',0,'00 USEC S/B LT 34 USEC',4
T1CMSG FCC '   C. HEAD LOAD DELAY TIME = '
T1CMSH FCC '   0000.00 MSEC S/B LT 200 MSEC',4
T1DMSG FCC '   D. WRITE PROTECT = '
T1DMSH FCC ' ',4
T1EMSG FCC '   E. INDEX PULSE COUNT = '
T1EMSH FCC '   S/B GT 2',4
T2MSG  FCC $0D,$0A,'II. DRIVE SPEED TEST',4
T2AMSG FCC '   PERIOD = '
T2AMSH FCC '0000.00 S/B 200+/-1 MSEC',4
OPTMSG FCC $0D,$0A,'  OPTIONS: 1.RETRY 2.CONTINUE 3.EXIT
',4
T3MSG  FCC $0D,$0A,'III. INDEX PULSE WIDTH',4
T3AMSG FCC '   WIDTH = '
T3AMSH FCC '0000.00 MSEC S/B 2.5-5.5 MSEC',4
NUL    FCC  0,0,0
GTHN   FCC 'GT '
T4MSG  FCC $0D,$0A,'IV. TRACK 0 SWITCH TEST',4
T4AERR FCC 'TRK 0 SWITCH FAULT; SHOULD TOGGLE GT 5
TIMES',4
T4AM1  FCC '   A. TIME FROM TRACK 0 TO SWITCH = '
T4AM3  FCC '   0000.00 MSEC',$0D,$0A
T4BM1  FCC '   B. TIME FROM TRACK 1 TO SWITCH = '
T4BM3  FCC '   0000.00 MSEC',$0D,$0A
T4CM1  FCC '   C. SWITCH POINT(A/A+B) = '
T4CM2  FCC '0000',$00,'00% S/B 5-95%',4
RESMSG FCC '   BUSY STUCK HI AFTER RESTORE',4
T5MSG  FCC $0D,$0A,'V. TRACK STEP TIME TEST',4
T5MSH  FCC '   STEP TIME = '
T5MSI  FCC '0000.00 MSEC S/B LT 40 MSEC',4
T5EMSG FCC '   SEEK TO TRACK TIMEOUT',4
RATBL  FCC $0F,$A0,$07,$D0,$03,$E8,$01,$F4,$00,$FA
T6MSG  FCC $0D,$0A,'VI. DOUBLE DENSITY STATUS (E0X4)',4
T6MS1  FCC '   A. RESTORE USING INTRQ(D6) -   ',4
T6MS2  FCC '   B. READ SECTOR USING DRQ(D7) -   ',4
T6MS3  FCC '   C. NO. BYTES READ = '
T6MS3A FCC '0000',$00,'00 S/B 256 BYTES',4
INTDAT FCC '101'
NGO    FCC 'NG'
OK     FCC 'OK'
T7MSG  FCC $0D,$0A,'VII. RADIAL ALIGNMENT TEST',$0D,$0A
T7MSG1 FCC 'ENTER-1.CHECK ALL TRACKS     2.ALIGN
3.EXIT '
T7MSH1 FCC $0D,$0A,'       4.CHECK SINGLE TRACK  5.NEXT
TEST '
T7MSH2 FCC '6.PRINT SYMBOLS ',4
T7FORM FCC 'ENTER FORMAT-1.SSSD 2.DSSD 3.SSDD 4.DSDD ',4
T7SYTB FCC 'SYMBOL TABLE:',S0D,$0A,'   * - VALID 256 BYTE
SECTOR'
T7SYTC FCC $0D,$0A,'   . - VALID 128 BYTE SECTOR'
T7SYTD FCC $0D,$0A,'   T - TRACK NUMBER ERROR'
T7SYTE FCC $0D,$0A,'   C - ID CRC ERROR'
T7SYTF FCC S0D,$0A,'   N - NO DATA ON TRACK'
T7SYTG FCC $0D,$0A,'   S - MISSING SECTOR'
T7SYTH FCC $0D,$0A,'   A - NO DATA ADDR MARK'
T7SYTI FCC $0D,$0A,'   D - DATA CRC ERROR'
T7SYTJ FCC $0D,$0A,'   O - OVER 256 BYTES'
T7SYTK FCC $0D,$0A,'   B - BETWEEN 129-255 BYTES'
T7SYTL FCC $0D,$0A,'   U - UNDER 128 BYTES',4
T7TRK  FCC '   TRACK(00-39)? ',4
T7TNO  FCC 'TRACK 00  ',4
T7CMSG FCC $0D,$0A,'A. ADJUST STEPPER MOTOR CLOCKWISE
UNTIL'
T7CM5H FCC ' HALF OF THE SECTORS HAVE ERRORS',4
T7CMS1 FCC $0D,$0A,'B. ADJUST STEPPER MOTOR COUNTER-
CLOCKWISE'
T7CMS2 FCC ' UNTIL HALF OF THE SECTORS HAVE ERRORS',4
T7CMS3 FCC $0D,$0A,'C. ADJUST STEPPER MOTOR SO IT IS'
T7CMS4 FCC ' CENTERED BETWEEN POINTS A. AND B.',$0D,$0A
T7CMS5 FCC 'SWITCH POINT S/B '
T7CMS6 FCC '0000',$00,'00%',4
T7MS2  FCC $0D,$0A,'ENTER DISK TYPE',S0D,$0A,'   1. OP-
SYSTEM DISK'
T7MS2A FCC $0D,$0A,'   2. DIGITAL DIAGNOSTIC DISK ',4
T7MS3  FCC 'WHICH SIDE (0 OR 1)? ',4
T7MS4  FCC $0D,$0A,'A. REDUCTION OF SIGNAL STRENGTH
TESTS',$0D,$0A
T7MS5  FCC '   1. 5/12 AMPLITUDE - ALTERNATE SECTORS
OFFSET'
T7MS5A FCC ' BY 7 MILS',4
T7MS6  FCC '   2. 4/12 AMPLITUDE - ALTERNATE SECTORS
OFFSET BY 8 '
T7MS6A FCC 'MILS',4
T7MS7  FCC '   3. 3/12 AMPLITUDE - ALTERNATE SECTORS
OFFSET BY 9 '
T7MS7A FCC 'MILS',4
T7MS8  FCC $0D,$0A,'B. TRACK OFFSET IN MILLI-
INCHES(MILS)',4
T7MS9  FCC 'TRACK NO.(00,05,16,19,30,39)? ',4
T7MS10 FCC 'TRY  STEP DIRN=IN  STEP DIRN=OUT    LIMITS
GO/NG'
T7MS1Z FCC $0D,$0A
T7MS1A FCC '-  ————  ————  ————  —',4
T7MS11 FCC ' X.   ',4
T7MS12 FCC ' +XX.X ',4
T7MS13 FCC 'MILS   ',4
T7MS14 FCC '   GO',4
T7MS15 FCC $0D,$0A,'C. AZIMUTHAL HEAD ANGLE TEST',4
T7MS16 FCC '   HEAD ANGLE =',4
T7MS17 FCC 'MIN  S/B 0 +/- 12 MIN',4
T7MS18 FCC '-1.5 TO +1.5',4
T8PROM FCC '$',4
T8MSG  FCC $0D,$0A,'VIII. READ/WRITE SECTOR TEST',4
T8SYM2 FCC '0123456789ABCDEFNL',$0D,4
T8SYM  FCC '12MERWNFHB',4
T8OPS  FCC 'ENTER OPTION: 1-PATTERN 1  2-PATTERN 2'
T8OPS1 FCC '  M-MODIFY PATTERN  E-EXAMINE
PATTERN',$0D,$0A,'R-READ'
T8OPS2 FCC ' SECTOR  W-WRITE SECTOR  N-NEXT SECTOR  F-
FLEX'
T8OPS3 FCC ' H-HELP  B-BACK TO TEST VII',4
T8MSG1 FCC 'BYTE NO.(00-FF)? ',4
T8MSG2 FCC 'CHANGE ',4
T8MSG3 FCC 'OPTIONS: N-NEXT BYTE  L-LAST BYTE  RTN-
EXIT',4
T8MSG4 FCC 'BYTE 00 = 00 ',4
T8MSG5 FCC 'BYTE:   #0 #1 #2 #3 #4 #5 #6 #7 #8 #9 #A'
T8MSH5 FCC '  #B #C #D #E #F   ASCII',4
T8MSG6 FCC 'X0-XF   00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00'
T8MSH6 FCC '   AAAAAAAAAAAAAAAA',4
T8MSG7 FCC S0D,$0A,'   SECTOR(01-36)? ',4
T8MSG8 FCC 'SECTOR READ ERROR',4
T8MSG9 FCC 'WRITE PROTECT ON',4
T8MS10 FCC 'WRITE ERROR',4
T8MS11 FCC 'SECTOR VERIFIED',4
T8MS12 FCC 'TRACK 00  SECTOR 00',4
T8MS13 FCC 'ARE YOU SURE(Y/N)? ',4
T8MS14 FCC 'SECTOR OUT OF RANGE',4
BFULL  FCC $0D,$0A,'BUFFER  FULL-SAVING',4
```

```
*        ************************
*        * PROGRAM SUBROUTINES *
*        ************************

*INCHNE-INPUT CHARACTER, NO ECHO
INCHNE LDA  TRMSTA  TERMINAL STATUS
       BITA #$01    DATA?
       BEQ  INCHNE  NO
       LDA  TRMDAT  GET CHARACTER
       ANDA #$7F
       RTS
INCH   LDB  TRMSTA
       BITB #$01
       BEQ  INCH
       LDA  TRMDAT
       ANDA #$7F
       JSR  OUTCH
       RTS
OUTCH  PSHS B,Y
       LDB  PRTFLG
       BEQ  OUTCH1
       LDY  PADDR
       CMPA #$0A
       BEQ  OUTCH1
       CMPY MEMEND  BUFF FULL?
       BNE  OUTCH0
       PULS B,Y,X
       LDX  #BFULL
       JSR  PSTR
       JMP  EXTO
OUTCH0 STA  ,Y+
       STY  PADDR
OUTCH1 LDB  TRMSTA
       BITB #$02
       BEQ  OUTCH1
       STA  TRMDAT
       PULS B,Y,PC
*FILCHK-PUTS FILE NAME INTS FCB
FILCHK LDX  LNPNTR  SAV FIL NAM
       STX  SAVLNP
       LDX  #FCB
       JSR  GETFIL
       BCS  FILERR
       LDA  #1      FCB<.TXT
       JSR  SETEXT
       INC  PRTFLG
       RTS
*
FILERR PULS X
       JSR  RPTERR
       JSR  FMSCLS
       JMP  WARMS
*CHKIN-CHECK CHARICTER IS IN RANGE,ECHO
CHKIN  STA  HI+1  UPPER LIMIT
       STB  LO+1  LOWER LIMIT
CHK1   JSR  INCHNE  GET CHARACTER
LO     CMPA #$F0    COMPARE UPPER
       BLT  CHKER
HI     CMPA #$0D    COMPARE LOWER
       BGT  CHKER
       BRA  CHKOK
CHKER  LDA  #$07    BELL
       JSR  OUTCH
       BRA  CHK1
CHKOK  JSR  OUTCH   OUTPUT CHARACTER
       RTS
*INCMP-INPUT CHAR AND COMPARE TO CHARS STARTING AT (X)
INCMP  CLR  TEMP    CHR POINTER
       JSR  INCHNE  GET CHAR
INCMP1 CMPA ,X      CHAR?
       BEQ  INCMP3
       INC  TEMP
       LDB  ,X+     EOF?
       CMPB #$04
       BNE  INCMP1  LOOP
INCMP2 LDA  #$07    BELL
INCMP3 LDB  TEMP
       JSR  OUTCH   PRINT
       RTS
*HEXCON-CONVERTS "A" REG TO ASCII
*MSB IN M(U),INC U: LSB IN M(U),INC U
HEXCON PSHS A       STORE BYTE
       ASRA         GET MSN
       ASRA
       ASRA
       ASRA
       BSR  HEXPUT  CONV TO ASCII
       STA  ,U+     STORE MSB CHAR
       PULS A       READ BYTE
       BSR  HEXPUT  CONV TO ASCII
       STA  ,U+     STORE LSB CHAR
       RTS
HEXPUT ANDA #$0F
       ADDA #$30
       CMPA #$39
       BLS  OP1
       ADDA #$07    CONV TO LETTER
OP1    RTS
*BDCON-BINARY TO DECIMAL CONV.
*X=BINARY MSB,U=DECIMAL LSB
*B=NO. OF BINARY BYTES ON ENTRY
*B=NO. OF DECIMAL BYTES ON RETURN
BDCON  BRA  BDS1
BBYTES RMB  1       NO OF BINARY BYTES
DLSB   RMB  2       LOCATION OF BCD LSB
BBCNTR RMB  1       L2 COUNTER
DBYTES RMB  1       NO OF DECIMAL BYTES
CCREG  RMB  1       TEMP CONDITION CODE
L3CNTR RMB  1       LOOP COUNTER
BDS1   LDA  #$01
       STA  DBYTES  SET NO DEC BYTES
       STB  BBYTES  SET NO BIN BYTES
       STU  DLSB    DECIMAL LSB POINTER
       CLR  ,U
L1     LDA  #$08    NO BITS PER BYTE
       STA  BBCNTR  SET COUNTER
L2     LDA  ,X      LOAD BIN BYTE
       LSLA         GET MS BIT
       STA  ,X      SAVE BIN BYTE
       LDU  DLSB    SET U TO DLSB
       LDB  #$00    CLEAR L3 CNTR
       STB  L3CNTR
L3     LDA  ,U      GET DEC BYTE
       ADCA ,U      C+2(DEC BYTE)
       TFR  CC,B    STORE CARRY
       STB  CCREG
       DAA          DEC ADJUST
       STA  ,U      STORE RESULT
       TFR  CC,B
       ORB  CCREG   ANY CARRIES?
       LEAU -1,U    POINT TO NEXT DBYTE
       INC  L3CNTR  INC COUNTER
       LDA  L3CNTR
       CMPA DBYTES  DECIMAL MS BYTE?
       BNE  OUT3
       ANDB #$01    CARRY?
       BEQ  L2END   NO
       INC  DBYTES  NEED MORE SPACE
       CLR  ,U      CLEAR SPACE
OUT3   TFR  B,CC    RESTORE CARRY
       BRA  L3
L2END  DEC  BBCNTR  LOOP COUNTER
       BNE  L2
L3END  DEC  BBYTES  GET NEXT BIN BYTE
       BEQ  FIN
       LEAX 1,X
```

```
       BRA   L1
FIN    LDB   DBYTES
       RTS
*DBCON-DECIMAL TO BINARY CONV.
*X=DECIMAL MSB, U=BIN LSB
*B=NO OF DEC BYTE-ON ENTRY
*B=NO OF BIN BYTES ON RETURN
DBCON  LDA   #$01    INIT BIN BYTES
       STA   BBYTES
       STB   DBYTES  NO DEC BYTES
       STU   BLSB
       STX   DMSB
DBL1   LDA   #$08    NO OF BITS
       STA   BBCNTR
DBL2   ANDCC #$00    CLR CARRY
       LDB   DBYTES
DBL3   JSR   DBSHR   RT SHFT
       LDA   ,U      GET BIN NO
       RORA
       STA   ,U
       LDX   DMSB
       DEC   BBCNTR
       BNE   DBL2
       NOP
DBL4   LEAX  1,X     INC X
       STX   DMSB
       DEC   DBYTES  NEXT BYTE?
       BEQ   DBOUT   FINISH
       LDA   ,X      GET BYTE
       BEQ   DBL4    ZERO?
       LEAU  -1,U    NEXT BIN BYTE
       INC   BBYTES
       BRA   DBL1    SHIFT DEC
DBOUT  LDB   BBYTES
       RTS
*DECADJ-IF BCD CHR GE 8, ADD 3
DECADJ PSHS  A
       ANDA  #$0F    LS ND
       CMPA  #$08
       BLO   DCAS1
       SUBA  #$03
DCAS1  STA   TEMP
       PULS  A
       ANDA  #$FD    MS NO
       CMPA  #$80
       BLO   DCAS2
       SUBA  #$30
DCAS2  ADDA  TEMP
       RTS
*DBSHR-ROTATE LS BIT INTO CARRY
DBSHR  LDA   ,X      GET DEC BYTE
       RORA
       PSHS  CC      CARRY
       JSR   DECADJ
       STA   ,X+     RESTORE DEC
       PULS  CC
       DECB          BIT CNTR
       BNE   DBSHR
       RTS
*GETDEC-TAKES TWO KEYBD INPUTS,CONV TO BIN # IN A
GETDEC LDD   #$3930  GET MSB
       JSR   CHKIN
       ASLA          PUT IN MSB
       ASLA
       ASLA
       ASLA
       STA   TEMP
       LDD   #$3930  GET LSB
       JSR   CHKIN
       ANDA  #$0F    ZERO MSB
       ORA   TEMP
       STA   TEMP
       LDU   #TEMP2  BIN #
       LDX   #TEMP   DEC #
       LDB   #$01    CONV 1 BYTE
       JSR   DBCON
       LDA   TEMP2
       RTS
*XFER-TRANSFER DATA; A=# BYTES,
*X=START OF SOURCE DATA,U=START OF DESTINATION
XFER   LDB   ,X+     GET CHAR
       STB   ,U+     STORE CHAR
       DECA          COUNTER
       BNE   XFER
       RTS
*TIMER-20USEC TIMING LOOP TO COUNT UNTIL
*BIT IN 'A' IS LO (B=27) OR HI (B=26)
*RESULTS STORED AT COUNT+1
TIMER  STA   BIT+1   TIMED BIT LOCN
       STB   BRHILO  TIMEOUT ON 1/0
       LDX   #$0001  INIT COUNTER
TCMD   LDA   #$0F    RESTORE
       STA   8,Y     CMD REG
TLOOP  LDB   8,Y     STATUS
BIT    BITB  #$00    CHK BIT
BRHILO BRA   TIMOUT  COMPARE
       NOP           DELAY
       LEAX  1,X     INC CNTR
       BNE   TLOOP   LOOP
MAXTIM LEAX  -1,X    REMOVE OFFSET
       STX   COUNT+1 STORE RESULT
       LDX   #GTHN   GET UPDATE
       BRA   TC1     CONTINUE
TIMOUT LEAX  -1,X    REMOVE OFFSET
       STX   COUNT+1 STORE RESULT
       LDX   #NUL    GET UPDATE
TC1    LDA   #$03    NO OF BYTES
       JSR   XFER    UPDATE MSG
       LDA   #$D0    INTERRUPT CMD
       STA   8,Y     CMD REG
       JSR   DELTS   1/10 SEC DEL
       RTS
*UPDMSG-UPDATE MESSAGE; TAKES DATA IN
*COUNT,CONVERTS TO DECIMAL DATA IN MSG
UPDMSG PSHS  U       TEMP STORE U
       LDA   #$00    CLR SCRATCH
       STA   COUNT   BIN MS BYTE
       STA   DIGNO   DECIMAL #
       STA   DIGNO+1
       ANDCC #$00    CLR CARRY
       LDX   #COUNT  BIN# SETUP
       LDU   #DIGNO+2 DEC# SETUP
       LDB   #$03    NO OF BYTES
       JSR   BDCON   CONVERT
       PULS  U       RESTORE MSG PNTR
       LDA   DIGNO   GET MSB
       JSR   HEXCON  CON+UPDATE
       LDA   DIGNO+1 GET NEXT BYTE
       JSR   HEXCON  CON+UPDATE
       LEAU  1,U     SKIP DEC PT
       LDA   DIGNO+2 GET LAST BYTE
       JSR   HEXCON  CON+UPDATE
       RTS
*CLKCMP-COMPENSATION FOR CLOCK RATE
CLKCMP LDA   #$00    CLR SCRATCH
       STA   COUNT   CLR MS BYTE
       LDA   CLKFLG
       BNE   CLKC1   SKIP IF 2MHZ
       ANDCC #$00    CLR CARRY
       ROL   COUNT+2 CNT:=2(CNT)
       ROL   COUNT+1
       ROL   COUNT
CLKC1  RTS
*PASFAL-CHECK TEST LIMITS, SET ERRFLG IF NG
PASFAL CMPD  COUNT+1 HI LIM OK?
       BHS   PSF1    OK
```

```
        INC   ERRFLG  ERR
PSF1    LDD   LIMT    GET LO LIMIT
        CMPD  COUNT+1 LO LIM OK?
        BLS   PSF2    OK
        INC   ERRFLG  ERR
PSF2    LDA   COUNT   MSB=0?
        BEQ   PSF3    OK
        INC   ERRFLG  ERR
PSF3    RTS
*TESTOK-CHECK IF TEST GO, OPTIONS IF NOGO
TESTOK  JSR   PSTRNG  PRINT
        LDA   ERRFLG  ERROR?
        BEQ   OKOUT
        JSR   OPTION
OKOUT   RTS
*DELAY-56USEC DELAY
DELAY   LBSR  DEL28
DEL28   LBSR  DEL14
DEL14   LBSR  DEL3
DEL3    RTS
*DELHS-HALF SECOND DELAY
DELHS   PSHS  D
        JSR   DELTS
        JSR   DELTS
        JSR   DELTS
        LDA   #$07
DELL1   LDB   8,Y     READY?
        BITB  #$80
        BEQ   DELOT
        JSR   DELTS
        DECA
        BNE   DELL1
DELOT   PULS  D
        RTS
*DELTS-TENTH SECOND DELAY
DELTS   PSHS  X       STORE X
        TST   CLKFLG
        BNE   D1T
        LDX   #$3333  INIT CNTR
        BRA   D2T
D1T     LDX   #$6666
D2T     LEAX  -1,X    DEC CNTR
        BNE   D2T     LOOP
        PULS  X       RESTORE X
        RTS
*PCRLF-CARRAGE RTN,LINE FEED
PCRLF   LDA   #$0A
        JSR   OUTCH
        LDA   #$0D
        JSR   OUTCH
        RTS
*PSTRNG-PRINT AND SET ERRFLG IF KEYBOARD ENTRY
PSTRNG  LDA   TRMSTA  TERMINAL STAT
        BITA  #$01    DATA?
        BEQ   PST0    NO
        INC   ERRFLG
PST0    JSR   PCRLF
PDATA   LDA   ,X+     GET CHAR
        CMPA  #$04    EOF?
        BEQ   PST2
        CMPA  #$00    NOP?
        BEQ   PDATA
        JSR   OUTCH   PRINT
        BRA   PDATA
PST2    RTS
*OPTION-TEST OPTIONS; U=RETRY ADDRESS
OPTION  LDX   #OPTMSG OPTIONS
        JSR   PDATA   PRINT
        LDD   #$3331  SET LIMITS
        JSR   CHKIN   GET CHAR
        ANDA  #$0F    CONV TO DEC
        SUBA  #$01    SUBT 1
        ASLA          MULT BY 2
        STA   OPTDIR+1 JUMP TABLE
OPTDIR  BRA   OPTDIR  JUMP
        BRA   OPT1
        BRA   OPT2
        BRA   OPT3
OPT1    PULS  X
        LDA   #$0D
        JSR   OUTCH
        LDB   8,Y
        JSR   DELHS
        LDX   RETRY
        JMP   ,X
OPT2    CLR   ERRFLG
        RTS
OPT3    PULS  X       CLR RTS ADDR
        JSR   PCRLF
*..PROGRAM EXIT
EXIT    JMP   EXT0
EXT0    TST   PRTFLG
        BEQ   EXT2
        LDX   #FCB
        LDY   SAVLNP FILE NAME
        STY   LNPNTR
        JSR   GETFIL
        BCS   OPTERR
        LDA   #1
        JSR   SETEXT
        LDA   #2      OPN TO WR
        STA   ,X
        JSR   FMS
        BEQ   EXT1
        LDA   1,X     IF FIL EXISTS(
        CMPA  #3
        BNE   OPTERR
        LDA   #12     DELETE FILE
        STA   0,X
        JSR   FMS
        BNE   OPTERR
        LDY   SAVLNP RESTOR FIL NAM
        STY   LNPNTR
        JSR   GETFIL
        BCS   OPTERR
        LDA   #1
        JSR   SETEXT FCB<.TXT
        LDA   #2      OPN TO WRITE)
        STA   0,X
        JSR   FMS
        BNE   OPTERR
EXT1    LDA   #$00
        STA   ,X
        LDY   #$3000
EXTLP   LDA   ,Y+
        JSR   FMS
        BNE   OPTERR
        CMPY  PADDR
        BLO   EXTLP
        LDA   #$0D
        STA   ,Y
        CMPY  PADDR
        BLS   EXTLP
        LDA   #4      CLOSE
        STA   ,X
        JSR   FMS
        BNE   OPTERR
EXT2    JMP   WARMS
OPTERR  JSR   RPTERR
        JSR   FMSCLS
        LDX   #SLTERR+22
        JSR   PSTR
        JSR   GETCHR
        CMPA  #'Y
        LBEQ  EXIT
        BRA   EXT2
```

```
*MUL3X1-COUNT:=CNTR X DVD
MUL3X1 LDU   #COUNT+3
       LDX   #DVD+3
       CLR   TEMP
       CLR   TEMP+1
       JSR   MUL1
       JSR   MUL1
       STA   COUNT
       RTS
MUL1   LDA   ,-X
       LDB   CNTR
       MUL
       ADDD  TEMP
       STB   ,-U
       STA   TEMP+1
       RTS
*MBDIV-MULTIBYTE DIVIDE DVD := DVD/DVR
*DATA FORMAT - MSB AT LOWEST ADDR
*CARRY=1 FOR ERROR, =0 FOR VALID RESULT
*SET 'LENGTH' TO NUMBER OF DATA BYTES
*RMDPTR POINTS TO LSB OF REMAINDER.
MBDIV  PSHS  X,Y,U
       LDA   LENGTH
       BNE   DINIT
       LBRA  EREXIT
*LCOUNT := 8(LENGTH)+1
DINIT  LDB   #$08
       MUL
       ADDD  #$0001
       STD   LCOUNT
*CLEAR ARRAYS & SET LSB POINTERS
       LDA   LENGTH
       LDY   #HIDE1
       LDU   #HIDE2
ZAIL   CLR   ,Y+
       CLR   ,U+
       DECA
       BNE   ZAIL
       STY   RMDPTR
       STU   TMPPTR
*SET LSB POINTERS DVRPTR, DVDPTR
       LDD   #DVD
       A0DB  LENGTH
       ADCA  #$00
       STD   DVDPTR
       LDD   #DVR
       ADDB  LENGTH
       ADCA  #$00
       STD   DVRPTR
*DIVISOR=0?
       LDA   LENGTH
       LDX   DVRPTR
CHKL1  LDB   ,-X
       BNE   DIV1
       DECA
       BNE   CHKL1
       LBRA  EREXIT
*DIVIDE USING TRIAL SUBTRACTION
DIV1   ANDCC #$FE   CLR C
DLOOP  LDA   LENGTH
       LDX   DVDPTR
SLLP1  LDB   ,-X
       ROLB
       STB   ,X
       DECA
       BNE   SLLP1
*DECR BIT CNTR AND EXIT IF DONE
*NO CHANGE TO CARRY
       LDX   LCOUNT
       LEAX  -1,X
       BEQ   OKEXIT
       STX   LCOUNT
*SHIFT THE CARRY INTO THE LS BIT OF THE
*UPPER DIVIDEND
       LDA   LENGTH
       LDX   RMDPTR
SLLP2  LDB   ,-X
       ROLB
       STB   ,X
       DECA
       BNE   SLLP2
*SUBTRACT DVR FROM REMAINDER PLACING THE
*DIFFERENCE INTO TEMP STORAGE
       LDA   LENGTH
       ANDCC #$FE  CLR C
       LDX   DVRPTR
SUBLP  LDB   ,-Y
       SBCB  ,-X
       STB   ,-U
       DECA
       BNE   SUBLP
       TFR   CC,A
       COMA
       TFR   A,CC
*IF CARRY =0, SUBTRACT NOT SUCCESSFULL,
*NEXT BIT OF QUOTIENT IS 0.
*IF CARRY=1, SUBTRACTION SUCCESSFULL,
*NEXT BIT OF QUOTIENT IS 1,
*SET HDEPTR TO NEW REMAINDER.
       LDY   RMDPTR
       LDU   TMPPTR
       BCC   DLOOP
       EXG   Y,U
       STY   RMDPTR
       STU   TMPPTR
       LBRA  DLOOP
OKEXIT ANDCC #$FE  CLR C
       BRA   DEXIT
EREXIT ORCC  #$01  SET C
DEXIT  PULS  X,Y,U,PC
*RESTOR-RETURN TO TRACK 0
RESTOR LDA   #$03     SLOW RESTORE
       LDX   #$0000   5 SEC TIMOUT
       STA   8,Y      CMD REG
RESLP  JSR   DELAY    28 USEC
       LDB   8,Y      STATUS
       BITB  #$01     BUSY?
       BEQ   RESOUT   NOT BUSY
       LEAX  1,X      COUNT
       BNE   RESLP    BUSY LP
RESERR LDX   RESMSG   TIMEOUT
       JSR   PSTRNG   PRINT
       JSR   OPTION   OPER OPS
RESOUT RTS
*RDDATA-READ ONE SECTOR OF DATA
RDDATA LDB   TRK      TRACK=0?
       BEQ   RDDAT0   SINGLE DEN
       LDB   DNSFLG
       BEQ   RDDAT0
       ORA   #$02     DOUBLE DEN
RDDAT0 LDB   T6FLG    IF T6 USE E0X4
       BEQ   RDDAT1
       BRA   RDDAT2
RDDAT1 STA   8,Y      CMDREG
       JSR   DELAY
       BRA   RDD4
RDD3   LDA   11,Y     DATA REG
       STA   ,X+      STORE DATA
RDD4   LDB   8,Y      STATUS
       BITB  #$02     DRQ?
       BNE   RDD3
       BITB  #$01     BUSY?
       BNE   RDD4
       RTS
*RDDAT2-READ DATA USING E0X4 STATUS
RDDAT2 STA   8,Y      CMDREG
```

```
       JSR   DELAY
       BRA   RDD40
RDD30  LDA   11,Y     DATA REG
       STA   ,X+
RDD40  LDB   4,Y      STATUS
       BMI   RDD30    DRQ?
       BEQ   RDD40
       LDB   8,Y      STATUS
       RTS
*WRDATA-WRITE DATA TO SECTOR OF DISK
WRDATA LDB   TRK      TRK=0?
       BEQ   WRDAT0   SING DEN
       LDB   DNSFLG
       BEQ   WRDAT0
       ORA   #$02     DOUBL DEN
WRDAT0 LDB   T6FLG    IF T6=1 USE
       BEQ   WRDAT1   E0X8 STATUS
       BRA   WRDAT2
*WRDAT1-WRITE DATA USING E0X8 STATUS
WRDAT1 STA   8,Y      CMDREG
       JSR   DELAY
       BRA   WDD4
WDD3   STA   11,Y     DATA REG
WDD4   LDA   ,X+
WDD5   LDB   8,Y      STATUS
       BITB  #$02     DRQ?
       BNE   WDD3
       BITB  #$01     BUSY?
       BNE   WDD5
       RTS
*WRDAT2-WRITE DATA USING E0X4 STATUS
WRDAT2 STA   8,Y      WRITE CMD
       JSR   DELAY
       BRA   WDD40
WDD30  STA   11,Y     DATA REG
WDD40  LDA   ,X+
WDD50  LDB   4,Y      STATUS
       BMI   WDD30
       BEQ   WDD50
       LDB   8,Y
       RTS
*SEEK-UPDATES TRACK, SECTOR & SIDE (B=SEC,A=TRK)
SEEK   STB   10,Y     SEC REG
       JSR   DELAY
       CMPA  9,Y      TRK REG
       BEQ   SKSEC
SKTRK  STA   11,Y     DATA REG
       JSR   DELAY
       LDA   #$1B     SEEK TRK
       STA   8,Y      CMD REG
       JSR   DELAY
SKT1   LDA   8,Y      STATUS
       BITA  #$01     BUSY?
       BNE   SKT1     LOOP
SKSEC  LDA   DRVCMD
       TST   9,Y      TRK 0
       BEQ   SKSD
       TST   DNSFLG   DOUB DENS?
       BNE   SKDD
SKSD   CMPB  #$0A     SEC GT 10?
       BGT   SKSD1    SIDE 1
       BRA   SKSD0
SKDD   CMPB  #$12     SEC GT 18?
       BGT   SKSD1
SKSD0  ANDA  #$40     SIDE 0?
       BEQ   NOUP     NO UPDATE
       LDA   DRVCMD   SET TO 0
       ANDA  #$BF
       BRA   SKCH
SKSD1  ANDA  #$40     SIDE 1?
       BNE   NOUP
       LDA   DRVCMD   SET TO 1
       ORA   #$40
SKCH   STA   DRVCMD
       STA   4,Y      CMD DRV
       JSR   DELTS    .1 SEC DEL
NOUP   RTS
*SCRAMBLE-GENERATES TRK FROM CNTR
SCRAMB LDA   CNTR     GET TRK CNT
       LSRA           MOV D2 TO D0
       LSRA
       ANDA  #$01
       STA   TRK      TEMP STORE
       LDA   CNTR
       LSLA           MOV D0 TO D2
       LSLA
       ANDA  #$04
       ORA   TRK      COMBINE
       STA   TRK      TEMP STORE
       LDA   CNTR
       ANDA  #$FA     CLR D0,D2
       ORA   TRK      REVERSE D0,D2
       STA   TRK      IN TRK
       RTS
*CONTIN-CONTINUE AFTER MANUAL ACTION
CONTIN JSR   PSTRNG   PRINT
       LDX   #DSKMSH+1 HIT RTN
       JSR   PDATA
       JSR   INCHNE   WAIT
       LDB   8,Y      STATUS
       JSR   DELHS    SPEEDUP
       RTS
*ASCII-EVALUATES IF REG A IS ASCII, IF NO SET TO #'.
ASCII  CMPA  #$20     LO ASCII
       BLO   ASET
       CMPA  #$7F     HI ASCII
       BHI   ASET
       BRA   AOUT
ASET   LDA   #'.      PERIOD
AOUT   RTS
*RDSEC-READ TRACK 'TRK' SECTOR 'SEC'; 'A' = 'U IF UNDER
256 BYTES
*'O IF OVER 256 BYTES, '* VALID 256 BYTE SECTOR, '. VALID
128 BYTE SEC.
*'D IF STATUS ERROR
RDSEC  LDX   #$0000   DATA POINTER
       LDA   TRK      SET TRACK
       LDB   SEC      SET SECTOR
       JSR   SEEK     STEP TO TRK
       LDA   #$8C     READ SEC CMD
       JSR   RDDATA   READ SEC
RDSEC1 BITB  #$08     CRC ERR?
       BEQ   RDSEC2
       LDA   #'D
       BRA   RDSEC9   PRINT
RDSEC2 BITB  #$10     REC NOT FND
       BEQ   RDSEC3
       LDA   #'S
       BRA   RDSEC9
RDSEC3 CMPX  #$0100   256 BYTES?
       BEQ   RDSEC5   OK *
       BLO   RDSEC4   LOW
       LDA   #'O      OVER
       BRA   RDSEC9
RDSEC4 CMPX  #$0080   128 BYTES?
       BEQ   RDSEC6   OK .
       BLO   RDSC4A   UNDR
       LDA   #'B      BTWN
       BRA   RDSEC9
RDSC4A LDA   #'U      UNDER
       BRA   RDSEC9
RDSEC5 LDA   #'*
       BRA   RDSEC9
RDSEC6 LDA   #'.
RDSEC9 RTS
*RDTRK-OUTPUT TRACK # AND SECTOR STATUS
```

```
RDTRK  LDA   TRK      GET TRK #
       STA   TEMP2    TEMP STORE
       LDX   #TEMP2   ADDR
       LDU   #TEMP    DEC ADDR
       LDB   #$01     CONV 1 BYTE
       JSR   BDCON    BIN TO DEC
       LDA   TEMP
       LDU   #T7TNO+6 MSG ADDR
       JSR   HEXCON   UPDATE MSG
       LDX   #T7TNO   MSG
       JSR   PSTRNG   PRINT
       LDB   #$01     FIRST SEC
       STB   SEC
RDTR1  JSR   RDSEC    READ SECTOR
       CMPA  #'S      REC NOT FND?
       BEQ   RDADD    FIND ERR TYPE
       CMPA  #'*      OK?
       BEQ   RDDUT0
       CMPA  #'.      OK?
       BEQ   RDOUT0
       BRA   RDOUT
RDADD  CLR   TEMP     LOOP CNT
RDADD1 LDX   #$0000
       LDA   #$C4     READ ADDR
       JSR   RDDATA
RDADD2 BITB  #$10     REC NOT FOUND
       BEQ   RDADD3   NO DATA
       LDA   #'N      ON TRK
       BRA   RDOUT
RDADD3 LDX   #$0002   SEC NO
       LDA   ,X
       CMPA  SEC
       BEQ   RDADD5   ID CRC
       INC   TEMP
       LDA   TEMP     READ ALL SEC?
       CMPA  MAXSEC
       BNE   RDADD1   LOOP
RDADD4 LDA   #'S      NO SEC #
       BRA   RDOUT
RDADD5 BITB  #$08     CRC ERR?
       BNE   RDAD06   TRK ERR
       LDA   #'C
       BRA   RDDUT
RDA0D6 LDA   ,X       TRK #
       CMPA  TRK
       BEQ   RDADD7   DATA ADDR MARK
       LDA   #'T      TRK ERR
       BRA   RDOUT
RDADD7 LDA   #'A      DATA AM
RDOUT  INC   ERRFLG   ERROR
RDOUTD LDB   TRMSTA   KEYSTROKE?
       BITB  #$01
       BEQ   RDOUT1
       INC   ERRFLG
RDOUT1 JSR   OUTCH    PRINT
       INC   SEC
       LDA   TRK      TRX 0?
       BNE   RDOUT2   NO
       LDB   FORMAT   DISK FFMT
       ANDB  #$01     SET TO SD
       LDX   #T7SECT  GET MAX SEC
       LDA   B,X
       BRA   RDOUT3
RDOUT2 LDA   MAXSEC
RDOUT3 CMPA  SEC      LOOP?
       LBHS  RDTR1
       RTS
*STPRTE-STEPS IN TO TRK0 FOR TIME=20USEC(X)
*       AND THEN CLEARS CONTROLLER
STPRTE LDX   STPSIZ   SET DELAY
       LDA   #$0F     RESTORE CMD
       STA   0,Y      CMD REG
DEL20U LBRN  DEL20U   DELAY
       LBRN  DEL20U
       NOP
       LEAX  -1,X     COUNT
       BNE   DEL20U
       LDA   #$D0     INTRPT
       STA   0,Y
       RTS
*OFFSET-FIND THE OFFSET OF THE HEAD WITH RESPECT TO THE
CAL DISK
*CENTERLINE.  RSLTN CONTAINS THE STEPSIZE OF THE OFFSET
QUANTITY
*IN MILS OR MINUTES.
OFFSET LDA   #$00      INIT VARS
       STA   SEC
       STA   CNTR      OFFSET CNTR
       STA   PCNT      +OFFSET
       STA   NCNT      -OFFSET
OFFHI  INC   SEC
       JSR   RDSEC     READ SEC
       CMPA  #'.       CORRECT?
       BEQ   OFFHI1
       CMPA  #'*       CORRECT?
       BNE   OFFLO
OFFHI1 LDA   CNTR      IF CORRECT
       STA   PCNT
OFFLO  INC   SEC
       JSR   RDSEC     READ SECTOR
       CMPA  #'.       CORRECT?
       BEQ   OFFLO1
       CMPA  #'*       CORRECT?
       BNE   OFFEX
OFFLO1 LDA   CNTR      IF CORRECT
       STA   NCNT
OFFEX  INC   CNTR      ALL SECTORS?
       LDA   CNTR
       CMPA  #$08
       BNE   OFFHI
       LDB   NCNT      NCNT:=-NCNT
       NEGB
       STB   NCNT
       LDA   PCNT      READ ALL +OFFSETS?
       CMPA  #$07
       BNE   OFF1
       ADDB  #$0C      PCNT:=NCNT+12
       STB   PCNT
       BRA   OFF2
OFF1   CMPB  #$F9      READ ALL -OFFSETS?
       BNE   DFF2
       SUBA  #$0C      NCNT:=PCNT-12
       STA   NCNT
OFF2   LDB   PCNT      B:=ABS(PCNT-NCNT)
       ADDB  NCNT
       BMI   OFF3      NEG OFFSET?
       LDA   #'+
       BRA   OFF4
OFF3   LDA   #'-
       NEGB
OFF4   STB   TEMP3
       STA   T7MS12+1  STORE SIGN
       LDA   RSLTN     COUNT:=B*RSLTN
       MUL
       STD   COUNT     SET LS DIGIT
       LSR   COUNT     OF RESULT
       ROR   COUNT+1
       BCC   OFF5
       LDA   #'5
       BRA   OFF6
OFF5   LDA   #'0
OFF6   STA   T7MS12+5
       LDX   #COUNT    SET MS 2 DIGITS
       LDU   #DIGNO+1  OF RESULT
       LDB   #$02
       JSR   BDCON
```

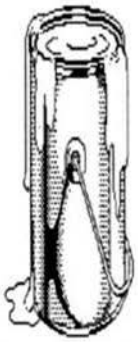
```
       LDU   #T7MS12+2
       LDA   DIGNO+1  CONV DIGIT
       JSR   HEXCON
       LDX   #T7MS12
       JSR   PDATA    PRINT
       RTS

*        ***************
*        * MAIN PROGRAM *
*        ***************

START  JSR   NXTCH    DRIVE NO?
       STA   TEMP
       LDX   #TITLE
       JSR   PSTRNG
       LDA   TEMP     DRIVE NO.?
       CMPA  #$0D     IN CMD LINE?
       BEQ   PROMPT   NO DATA
       CMPA  #$0A
       BEQ   PROMPT
       CMPA  #$30     VALID ENTRY
       BLT   STRT0
       CMPA  #$33
       BGT   STRT0
       BRA   DEFLT
STRT0  CMPA  #'>
       BEQ   STRT1
USGER  LDX   #USGERR
       JSR   PSTRNG
       JMP   WARMS
STRT1  JSR   FILCHK
       BRA   PROMPT
DEFLT  JSR   NXTCH
       CMPA  #$20
       BEQ   DEFLT
       CMPA  #'>
       BNE   DEF1
       JSR   FILCHK
DEF1   LDY   DRVADDR INIT SLOT
       LDA   TEMP
       STA   DSKMSH
       ANDA  #$0F
       STA   DRVCMD
       LDX   #STRT3
       STX   RETRY
*      LDA   #$00     0=1M,1=2MH2
*      STA   CLKFLG
       CLR   ERRFLG
       LBRA  T1
PROMPT LDX   #SLTMSG I/O SLOT?
       JSR   PSTRNG  PRINT
       LDD   #$3730  SET HI/LO
       JSR   CHKIN   GET CHAR
       ANDA  #$0F    ASCII TO BIN
       ASLA
       ASLA
       ASLA
       ASLA
       STA   DRVADD+1 SET CONT. ADDR
       LDY   DRVADD   I/O ADDR
       LDA   #$AA     TEST DATA
       STA   11,Y     DATA REG
       JSR   DELAY
       LDA   11,Y     CONTROLLER?
       CMPA  #$AA
       BEQ   PROMP2
       LDX   #SLTERR
       JSR   PSTRNG
       JSR   INCH
       CMPA  #'Y
       BEQ   PROMPT
       LBRA  EXIT
PROMP2 LDX   #DRVMSG DRIVE NO?
       JSR   PSTRNG
       LDD   #$3330
       JSR   CHKIN   GET CHAR
STRT2  STA   DSKMSH  STORE DRIVENO
       ANDA  #$DF
       STA   DRVCMD
PROMP3 LDX   #CLKMSG CLOCK RATE?
       JSR   PSTRNG PRINT
       LDD   #$3231  LIMITS
       JSR   CHKIN
       ANDA  #$0F    ASCII->BIN
       LSRA          A:=A-1
       STA   CLKFLG
STRT3  CLR   ERRFLG  CLEAR ERRORS
       LDX   #HLTMSG HALT AFTER TST
       JSR   PSTRNG
       JSR   INCH
       CMPA  #'Y
       BNE   NOHLT
       INC   ERRFLG  SET ERROR
       LDX   #HLTMSG
       STX   OPT2+1
       BRA   NOHLT1
NOHLT  LDX   #ERRFLG
       STX   OPT2+1
NOHLT1 LDX   #DSKMS1 INSERT DISK
       JSR   PSTRNG  PRINT
       JSR   INCHNE  WAIT
       LDX   #PROMP3 RETEST ADDR
       STX   RETRY   STORE
*      TEST 1 - DRIVE MDTOR STATUS(E0X0)
T1     LDX   #T1MSG  DRV MOT STATUS
       JSR   PSTRNG  PRINT
       LDA   DRVCMD  GET CMD
       STA   4,Y     DRVREG
T1A    LDU   #T1AMSH READY DEL TIME
       LDD   #$0027  TIME D7=LO
       JSR   TIMER   MEAS TIME
       JSR   CLKCMP  CLK COMPENS
       LDD   #$0000  LO=0SEC
       STD   LIMT
       LDD   #$F230  HI=.62SEC
       JSR   PASFAL  LIM OK?
       JSR   UPDMSG  UPDATE MSG
       LDX   #T1AMSG GET MSG
       JSR   TESTOK  PRINT
T1B    LDU   #T1BMSH BUSY RESP TIME
       LDA   #$04    INIT TIMER
       STA   CNTR    STORE
       LDB   #$18    INIT BRANCH
       STB   T1B2+1  STORE
T1B1   LDA   #$00    RESTORE NO VER
       STA   0,Y     CMDREG
T1B2   BRA   T1B3    3U DELAY
       NOP           2U DELAY
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
       NOP
```

To Be Continued Next Month

Bit-Bucket



By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

MOTOROLA INC.

CONTACTS
Shernaz Daver
Cunningham Communication, Inc.
(408) 982-0400

Dean Mosley
Microprocessor Products Group
(512) 891-2839

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

HCL AMERICA ANNOUNCES 68030-BASED MINICOMPUTER

System Incorporates up to Six 68030 Processors and 68882 Math Coprocessors

AUSTIN, Texas, Feb. 6, 1989 — HCL America (Sunnyvale, Calif.), a subsidiary of India's largest computer manufacturer, today announced a multiprocessor minicomputer based on Motorola's 68030 (030) processor and 68882 (882) math coprocessor. The system, called M3000 series, uses up to six 25 MHz 030s each coupled with a 25 Mhz 882 and is targeted at office automation and on-line transaction processing.

HCL's announcement adds to a growing base of manufacturers offering 030-based products including Apple Computer, Apollo, Hewlett-Packard, NEC and Sony Microsystems. The 030 is the top-of-the-line microprocessor in Motorola's 68000 family that includes the 68000, 68008, 68010 and 68020. The 68000 family has generated the largest software base ($3 billion) and most established hardware base ($100 billion). The 030's compatibility with preceding members of the 68000 family protects manufacturers' original software investments by allowing for easy migration of software from one 68000 processor to another.

"The 030's versatility and price/performance advantage make it a natural choice for a wide range of markets," said Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "Our soon-to-be-announced 68040 will be a revolutionary product that supports our large 68000 installed base."

HCL America is a wholly owned subsidiary of HCL Ltd., India's largest computer company and leading supplier of Unix-based machines. HCL America, headquartered in Sunnyvale, Calif., markets Unix-based multiprocessor computer systems and software, as well as customized software services to OEM customers.

Motorola's $2.2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a part of Motorola Inc. It is the largest and broadest supplier of semiconductors in North America, with a balanced portfolio of over 50,000 devices.

SOFTWARE DEVELOPMENT SYSTEMS, INC

4248 BELLE AIRE LANE • DOWNERS GROVE, ILLINOIS 60515 • USA
PHONE: 1-312-971-8170 • FAX: 1-312-971-8513
Contact: James E. Challenger

February 21, 1989

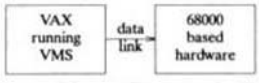
Mr. Larry Williams
Executive Editor
68 Micro Journal
Computer Publishing, Inc.
5900 Cassandra Smith Road
Hixson, Tennessee 37343

Dear Sir,

Thanks for letting us know about your journal and please excuse me for not getting back to you. Since your magazine is dedicated to the coverage of Motorola microprocessors, I would expect that your readers will be interested in the enclosed press release regarding our CrossCode C product, which is just now being introduced under the VAX VMS operating system. CrossCode C is an optimizing C compiler that runs on the VAX and generates ROMable code for all members of Motorola's 68000 family of microprocessors.

CrossCode C is used to develop a wide variety of 68000-based applications, including automotive control devices, aerospace and defense products, amusement devices, computer and communications equipment, consumer products, laser scanning equipment, and medical instruments.

CrossCode C allows our customers to develop code for these devices using the VAX, which provides a complete and friendly software development environment. Here's how it works:

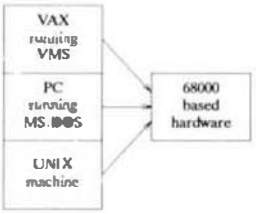


The programmer uses all the powerful and familiar tools on the VAX (i.e. terminals, printers, file system, text editor, etc.) to produce C source code for a program that will eventually run on the 68000 based hardware. CrossCode C then compiles this source code into an *object file* which represents the actual instruction codes to be executed by the 68000. This object file is then converted into a format that is suitable for transmission down a data link to the 68000 hardware, where the object code is executed. In a nutshell, that's how CrossCode C is used.

Is there anything really special about this product introduction? Yes there is. CrossCode C is now one of the very few 68000 C compilers that provides identical capabilities under three different operating system environments: VMS, UNIX, and MS-DOS. Not only that, but it allows users to move their source, object code, and object libraries to different types of machines without having to recompile or relink them. That's because CrossCode C's object files are independent of the machine on which they were generated.

For example, a CrossCode C user can take object files or libraries that were generated on a VAX, move them to a PC or a SUN and then use the PC or SUN as a development machine. People who have networks containing machines that support two or three different operating systems (VMS, UNIX or MS-DOS) can share their object files across the entire network without having to worry about what type of machine they happen to be using at the moment.

Any machine on the network can be used as a development station to generate code for a 68000 based application. Here's an example:



I hope this letter has given you enough information to decide whether our release will interest your readers. If you have any questions about CrossCode C or its significance to VMS users, please give me a call at 1-312-971-8170 and I'll gladly give you more information.

I've enclosed two black-and-white photos, but if you prefer a color photo I'll send you one. One enclosed photo is in landscape orientation, and the other is a portrait. If you plan to print the press release, please consider that your readers may want to see a picture of the product.

Thank you for your time and attention.

Sincerely,

David F. Ziffer, Vice President

ROMable 68000 C Compiler Now Available Under VAX VMS

Eight months after introducing its CrossCode C compiler under MS-DOS and UNIX, Software Development Systems, Inc. (SDSI) has announced that the compiler is now being made available under VAX VMS. CrossCode C is an optimizing C compiler that generates ROMable code for all 68000 family members, including the high-end 68020 and 68030 processors, the 68881 and 68882 floating point coprocessors, and the 68851 MMU. Users may write source code in any combination of C and assembly language, compile the code into object files, and then link the object files to form a complete load for any 68000 family processor.

SDSI found that many of the companies using VMS for embedded systems development were also using MS-DOS and UNIX based machines. Because of this, SDSI designed CrossCode C to run identically and to share source and object files across all three operating systems. For example, the CrossCode C linker on the VAX can link object files that were generated by the CrossCode C compiler under MS-DOS. This feature allows users to move their source code, object files and libraries from machine to machine without making any changes at all.

CrossCode C comes with a Motorola-compatible macro assembler, a linker, a librarian, and a C library containing the source to over 47 C functions. Also included is a universal downloader that communicates with EPROM programmers, emulators and target hardware. The downloader converts users' compiled C code into Motorola S-Records and a variety of other industry standard file formats. It also emits symbolic information to provide symbolic debugging capabilities on many popular emulators, including those made by Applied Microsystems, Atron, Huntsville Microsystems, MicroCASE, Northwest Instruments, Orion Instruments, Softaid, Sophia, and ZAX.

CrossCode C runs on all members of the VAX family, including the MicroVAX 2000, MicroVAX II, MicroVAX 3500/3600, VAX 750 and 780, VAX 6200, and VAX 8000 Series machines. Prices start at $1,595 on the MicroVAX 2000.

SOFTWARE DEVELOPMENT SYSTEMS, INC. was formed in 1984 to provide cross development software to people who build ROMable applications. Since then SDSI has provided cross compilers and cross assemblers to many hundreds of companies and their consultants. Software Development Systems, Inc. is dedicated entirely to the design, manufacture, and support of cross development software.

Continued from page 57 of March 89 issue.

DO - A FLEX-09 Batch File Processor

By: Dave Howland

```
        FCB     0
        FDB     PARON
        FCC     'THEN'
        FCB     0
        FDB     PARTHN
        FCB     0
*
* MESSAGES FOR RESIDENT PART OF 'DO'
*
CMDERM  FCC     'FLEX command error'
        FCB     4
BREAKM  FCC     'BREAK entered'
        FCB     4
ABORTM  FCC     'Batch file aborted'
        FCB     4
FILERM  FCC     'Illegal file type'
        FCB     4
PREFM   FCC     'Illegal parameter reference in batch file'
        FCB     4
CMDPEM  FCC     'Batch file command error'
        FCB     4
CONTNM  FCC     'Press any key to continue ... '
        FCB     4
NOLABM  FCC     'Label not found'
        FCB     4
INIFM   FCC     'ELSE or ENDIF not found'
        FCB     4
NOIFM   FCC     'ELSE or ENDIF without IF'
        FCB     4
MENDM   FCC     'MEMEND changed by batch file; DO memory not released'
        FCB     4
*
*****************************************************************************
*
* MAIN COMMAND LOOP
*
*****************************************************************************
*
RESID   JSR     PCRLF
RESID1  LEAX    READCH,PCR      SETUP INPUT VECTOR TO 'DO' CHAR INPUT
ROUTINE
        STX     INCH+1
        LBSR    TSTABT          TEST FOR ABORT FROM THE USER
        LBSR    READLN          READ NEXT LINE FROM COMMAND FILE
        LDA     LINBUF          IGNORE EMPTY LINES
        CMPA    #CR
        BEQ     RESID1
        BSR     TSTCMD          TEST FOR A 'DO' INTERNAL COMMAND
        BNE     RESID1
        JSR     DOCMND          IF NOT, CALL DOS
        STB     LSTERR,PCR      SAVE LAST DOS ERROR
        TST     ERRFLG,PCR      ONLY CHECK ERROR IF ERRFLG SET
        BEQ     RESID1
```

```
        FCC     'GOTO'
        FCB     0
        FDB     DOGOTO-CTABLE
        FCC     'IF'
        FCB     0
        FDB     DOIF-CTABLE
        FCC     'NOTE'
        FCB     0
        FDB     DONOTE-CTABLE
        FCC     'ON'
        FCB     0
        FDB     DOON-CTABLE
        FCC     'PAUSE'
        FCB     0
        FDB     DOPAUS-CTABLE
        FCC     'REM'
        FCB     0
        FDB     DOREM-CTABLE
        FCC     'SHIFT'
        FCB     0
        FDB     DOSHFT-CTABLE
        FCB     0
*
* PARAMETER TABLE
*
PTABLE  FCC     'CONTINUE'
        FCB     0
        FDB     PARCON
        FCC     'ELSE'
        FCB     0
        FDB     PARELS
        FCC     'ENDIF'
        FCB     0
        FDB     PARNDF
        FCC     'ERROR'
        FCB     0
        FDB     PARERR
        FCC     'EXIST'
        FCB     0
        FDB     PAREXS
        FCC     'GOTO'
        FCB     0
        FDB     PARGTO
        FCC     'IF'
        FCB     0
        FDB     PARIF
        FCC     'NOT'
        FCB     0
        FDB     PARNOT
        FCC     'OFF'
        FCB     0
        FDB     PAROFF
        FCC     'ON'
```

```
        LBNE    DSKERR          ERROR
FNOLB1  LBSR    READLN          READ LINE FROM COMMAND FILE
        LDA     LINBUF          IF NO LABEL THEN CONTINUE (UNTIL EOF)
        CMPA    #'@
        BNE     FNDLB1
        JSR     NXTCH           ELSE MOVE LINE POINTER TO START OF LABEL
        LDX     LABPTR,PCR
        LBSR    COMPAR          COMPARE LABELS
        BEQ     FNDLB1          CONTINUE IF NO MATCH
        LDA     SAVBCH,PCR      RESTORE ECHO FLAG
        STA     ECHOFL,PCR
        CLR     GOTOFL,PCR      CLEAR GOTO FLAG
        RTS                     ELSE RETURN
*
*
* PROCESS 'IF' COMMAND
*
DOIF    CLR     NOTFLG,PCR      CLEAR 'NOT' PARAMETER FLAG
        LBSR    CHKPAR          CHECK PARAMETER
        CMPB    #PARNOT         IF 'NOT'
        BNE     DOIF1
        COM     NOTFLG,PCR      THEN SET 'NT' PARAMETER FLAG
        LBSR    CHKPAR          AND GET NEXT PARAMETER
DOIF1   CMPB    #PARXST         IF 'EXIST'
        BNE     DOIF2
        LDX     MEMEND          SETUP TEMPORARY FCB
        LEAX    -F_LEN,X
        JSR     GETFIL
        BCS     DOERR           REPORT DOS ERROR, INVALID FILENAME
        LDA     #M_OPNR         TRY TO OPEN FILE FOR READ
        STA     F_FUNC,X
        JSR     FMS
        BEQ     DOIF1A
        LDA     F_ERR,X         IF ERROR = FILE NOT FOUND THEN FALSE
        CMPA    #E_FNF
        BEQ     DOFLSE
DOERR   LBRA    DSKERR          ELSE DISK ERROR
*
DOIF1A  LDA     #M_CLOS         IF NO ERROR, CLOSE FILE AND GOTO TRUE
        STA     F_FUNC,X
        JSR     FMS
        BNE     DOERR
        LBRA    DOTRUE
*
DOIF2   CMPB    #PARERR         ELSE IF 'ERROR'
        BNE     DOIF3
        TST     LSTERR,PCR      THEN IF LAST ERROR FLAG <> 0
        LBNE    DOTRUE          THEN TRUE
        BRA     DOFLSE          ELSE FALSE
*
DOIF3   LDY     LINPTR          ELSE LOOK FOR '='
        PSHS    Y
DOIF4   JSR     NXTCH
        CMPA    #CR             IF CR FOUND, HAVE PARAMETER ERROR
        LBEQ    CMDPER
        CMPA    #'=
        BNE     DOIF4
        LDY     LINPTR          CHANGE '=' TO SPACE
        LDA     #SPACE
        STA     ,-Y
        LEAY    1,Y             IF NEXT CHAR = DELIMITER
        LDA     0,Y
        CMPA    #SPACE
        BHI     DOIF5
        JSR     NXTCH           THEN GET NEXT CHAR
DOIF5   PULS    X               RESTORE POINTER TO FIRST STRING
        LDY     LINPTR          SAVE LINPTR
        LBSR    COMPAR          COMPARE STRINGS
        BNE     DOTRUE
        STY     LINPTR          RESTORE LINPTR
DOIF6   JSR     NXTCH           MOVE PAST SECOND STRING TO CHAR <= SPACE
        CMPA    #SPACE
        BHI     DOIF6
*
DOFLSE  TST     NOTFLG,PCR      FALSE; TEST 'NOT' FLAG
        BNE     DOTRU1
DOFLS1  LBSR    CHKPAR          DEFINITELY FALSE; CHECK PARAMETER
        CMPB    #PARTHN         IF NOT 'THEN' OR 'GOTO' HAVE ERROR
        BEQ     DOFLS2
        CMPB    #PARGTO
        LBNE    CMDPER
        RTS
*
DOFLS2  LDA     ECHOFL,PCR      IF 'THEN' SAVE CURRENT ECHO FLAG
        STA     SAVBCH,PCR
        CLR     ECHOFL,PCR      DISABLE ECHO
        INC     GOTOFL,PCR      SET GOTO FLAG TO DISABLE PARAMS
        CLR     IFLEVX,PCR      CLEAR BYPASS 'IF' LEVEL
DOFLS3  LBSR    READLN          READ NEXT COMMAND LINE
        LBSR    CHKPAR
        CMPB    #PARIF          IF 'IF'
        BNE     DOFLS4
        INC     IFLEVX,PCR      THEN INCREMENT BYPASS 'IF' LEVEL
        BRA     DOFLS3          CONTINUE
*
DOFLS4  CMPB    #PARNDF         IF 'ENDIF'
        BNE     DOFLS5
        DEC     IFLEVX,PCR      THEN DECREMENT BYPASS 'IF' LEVEL
        BPL     DOFLS3
        BRA     DOFLS6          EXIT LOOP IF < 0
*
DOFLS5  TST     IFLEVX,PCR      IF BYPASS 'IF' LEVEL > 0
        BNE     DOFLS3          THEN CONTINUE LOOP
        CMPB    #PARELS         IF 'ELSE'
        BNE     DOFLS3
        INC     IFLEVL,PCR      THEN INCREMENT 'IF' NESTING LEVEL
```

```
        TSTB                    IF DOS ERROR
        BEQ     RESID1
        LEAX    ERRLAB,PCR      THEN GOTO LABEL IN ERRLAB
        LBSR    FNDLAB
        BRA     RESID1
*
*****************************************************************************
*
* INTERNAL COMMAND PROCESSING ROUTINES
*
*****************************************************************************
*
* TEST FOR INTERNAL 'DO' COMMAND, AND PROCESS IT
* RETURN Z = 0 FOR INTERNAL COMMAND, Z = 1 OTHERWISE
*
TSTCMD  LDA     [LINPTR]        GET FIRST CHAR OF NEW COMMAND
        CMPA    #'!             IF COMMENT CHAR THEN PROCESS COMMENT
        BEQ     TSTCM1
        CMPA    #'@             IF LABEL CHAR THEN PROCESS AS A COMMENT
        BNE     TSTCM2
TSTCM1  LBSR    DOREM
        BRA     TSTCM3
*
TSTCM2  LBSR    CHKCMD          ELSE CHECK FOR AN INTERNAL 'DO' COMMAND
        BEQ     TSTCM4
        JSR     0,X             IF IT IS, CALL ROUTINE TO PROCESS IT
TSTCM3  LDB     #1              CLEAR ZERO FLAG TO INDICATE INTERNAL
ROUTINE
TSTCM4  RTS
*
*
* PROCESS 'ECHO' COMMAND
*
DOECHO  LBSR    CHKPAR          CHECK PARAMETER
        CMPB    #PARON          IF 'ON'
        BNE     DOECH1
        LDA     #$FF            THEN SET ECHO FLAG ON
        STA     ECHOFL,PCR
        RTS
*
DOECH1  CMPB    #PAROFF         ELSE IF NOT 'OFF'
        LBNE    CMDPER          THEN HAVE ERROR
        CLR     ECHOFL,PCR      ELSE SET ECHO FLAG OFF
        RTS
*
*
* PROCESS ELSE COMMAND
*
DOELSE  TST     IFLEVL,PCR      IF 'IF' NESTING LEVEL = 0
        BEQ     NIFERR          THEN HAVE ERROR
        LDA     ECHOFL,PCR      ELSE SAVE CURRENT ECHO FLAG
        STA     SAVECH,PCR
        CLR     ECHOFL,PCR      DISABLE ECHO
        INC     GOTOFL,PCR      SET GOTO FLAG TO DISABLE PARAMS
        CLR     IFLEVX,PCR      CLEAR BYPASS 'IF' LEVEL
DOELS1  LBSR    READLN          READ NEXT COMMAND LINE
        LBSR    CHKPAR
        CMPB    #PARIF          IF 'IF'
        BNE     DOELS2
        INC     IFLEVX,PCR      THEN INCREMENT BYPASS 'IF' LEVEL
DOELS2  CMPB    #PARNDF         CONTINUE UNTIL 'ENDIF'
        BNE     DOELS1
        DEC     IFLEVX,PCR      DECREMENT BYPASS 'IF' LEVEL
        BPL     DOELS1          CONTINUE IF >= 0
        CLR     GOTOFL,PCR      CLEAR GOTO FLAG
        LDA     SAVECH,PCR      RESTORE ECHO STATUS
        STA     ECHOFL,PCR
DOELS3  DEC     IFLEVL,PCR      DECREMENT 'IF' NESTING LEVEL
        RTS
*
*
* PROCESS ENDIF COMMAND
*
DOENDF  TST     IFLEVL,PCR      IF 'IF' NESTING LEVEL <> 0
        BNE     DOELS3          THEN DECREMENT 'IF' NESTING LEVEL
NIFERR  LEAX    NOIFM,PCR       ELSE REPORT MISSING 'IF' MESSAGE
        LBRA    PERROR
*
*
* PROCESS 'EXIT' COMMAND
*
DOEXIT  LBRA    EXIT
*
*
* PROCESS 'GOTO' COMMAND
*
DOGOTO  LEAX    LABEL,PCR       MOVE LABEL TO LABEL BUFFER
        LDB     #0
DOGOTD  JSR     NXTCH
        STA     ,X+
        CMPA    #SPACE
        BLS     DOGOT1
        DECB
        BNE     DOGOTD
        CLR     ,X
DOGOT1  LEAX    LABEL,PCR
*
FNDLAB  STX     LABPTR,PCR      SAVE LABEL POINTER
        LDA     ECHOFL,PCR      SAVE CURRENT ECHO FLAG
        STA     SAVECH,PCR
        CLR     ECHOFL,PCR      TURN ECHO OFF
        INC     GOTOFL,PCR      SET GOTO FLAG
        LEAX    DOFCB,PCR       REWIND TO START OF COMMAND FILE
        LDA     #M_RWND
        STA     F_FUNC,X
        JSR     FMS
```


To Be Concluded Next Month

CONTACTS
Bob Anundson
88open Consortium, Ltd.
(503) 682-5703

88OPEN CONSORTIUM PROMOTES OPEN STANDARDS FOR MOTOROLA 88000 RISC MICROPROCESSOR; DEMONSTRATES "NEW WAY OF DOING BUSINESS"

The 88open Consortium is a group of leading hardware and software vendors, formally chartered in April 1988 to establish and promote the industry-wide adoption of open standards for products using the Motorola 88000 RISC (Reduced Instruction Set Computing) microprocessor architecture. The members of 88open are leading the way to a new way of doing business in the computer industry — demonstrating that vendors who compete vigorously against one another in the marketplace can benefit from dedicating resources to a cooperative effort to achieve mutual success.

The 88open Mission

The mission of the 88open Consortium reflects the common belief of its membership that the Motorola 88000 microprocessor RISC architecture offers a superior platform for the development of computer systems and software. The timely implementation of standards for hardware and software interfaces to the 88000, prior to the development of a substantial body of systems and software, will achieve clear benefits for both vendors and users of 88000-based products.

The existence of standards will reduce the complexity and cost of developing operating systems, compilers, applications software, and peripheral hardware. Moreover, having invested once in designing for a standard environment, vendors can then invest more of their development resources in improving and differentiating their products. The elaborate porting activities typically needed to move software from one vendor's system to another's will be virtually eliminated. The resulting cost savings will benefit vendors and users alike.

88open welcomes participation in its standards-setting activities from all interested parties, including the academic and user communities. All standards developed by the Consortium will be shared openly and equally. 88open members will automatically receive copies of standards-related documents, and non-members may receive them for a nominal fee.

The members of the Consortium believe product development based on standards will accelerate market acceptance of the 88000 through earlier availability of a broader selection of innovative systems and software at reasonable prices. Customers as well as vendors will be more willing to invest in products based on open standards if they are confident of not being penalized for early commitment to a young technology.

The 88open Charter

The 88open Consortium charter aims to bring about the market success of the 88000 RISC architecture through a variety of programs, beginning with educational and marketing activities to promote the 88000 product family as the most complete and viable RISC processor platform available to the market.

As a part of its mission to develop open standards and encourage their use, the Consortium will participate in standards-setting activities worldwide, wherever appropriate. The Consortium also will speak as a collective voice to influence future development of the 88000 product family.

The Consortium may negotiate with hardware and software vendors to produce 88000-related products for the benefit of 88open members. In addition, members may cooperate to develop products that meet mutual needs, through collaboration between members or through research and development sponsored by the Consortium. It is expected that Consortium members will benefit from more favorable access to such products.

Early Success: The 88000 Binary Compatibility Standard

Only three months after being formally chartered, the Consortium published the *Trial Use Standard* document for the 88000 Binary Compatibility Standard (BCS). The first BCS published for any RISC microprocessor, this standard will allow software written to its specifications to run on any BCS-compliant 88000-based system.

The 88000 BCS is compatible with such widely accepted industry standards as POSIX, X/Open, and the AT&T UNIX System Interface Definition. It supports 90 percent of the specifications developed for the proposed Motorola 68000 BCS, simplifying the portability of applications from one architecture to the other. In addition, the 88000 BCS is implemented at the system call level, and is therefore applicable to any variant of the UNIX operating system or to a non-UNIX operating system.

The 88000 BCS is a subset of the proposed AT&T application binary interface (ABI), a more complete interface standard that will eventually specify extension features such as shared library support, network interface and the "look and feel" of the windowing system. AT&T has indicated informally its intention to use the 88000 BCS as the basis for an 88000 ABI.

The 88open Consortium will work with AT&T to complete the 88000 ABI, and will continue to develop certification tools to verify that application programs conform to the BCS standard.

The 88000 BCS exemplifies the manner in which strategic cooperation among enlightened companies can yield substantive benefits, not only to the direct participants, but also to vendors and customers throughout the industry. There is no doubt that the establishment of this standard so near the introduction of the 88000 architecture will facilitate its rapid acceptance and motivate a variety of developers to exploit its powerful capabilities.

Organization

The Consortium is organized as a not-for-profit member organization. It is governed by a Board of Directors representing member companies, with three directors from hardware vendors, two from software vendors, one from the academic community, one from the membership at large, and one from Motorola.

Day-to-day operations are run by a full time executive director, who also serves on the Board of Directors. Bob Anundson, formerly of Tektronix, Inc., is executive director. He is assisted by a small staff.

The work of the Consortium is carried on through member volunteer committees. The Technical Committee is charged with proposing and developing definitions, standards and extensions relating to the use of the 88000 architecture and its use in hardware and software; this committee produced the 88000 BCS.

The Business Committee deals with business issues relating to the Consortium, and has broad responsibility for the promotion of the 88000 product family in the marketplace. The World Strategies Committee will focus on the extension of the Consortium's activities and influence worldwide. 88open Consortium offices and staff are planned for Europe and the Pacific Rim.

Corporate members have full voting rights, and thus participate in setting the goals and determining the activities of the Consortium. Non-voting memberships are offered to individuals and non-profit organizations who are interested in the goals of the Consortium.

For more information, contact 88open Consortium, Ltd., 8560 SW Salish Lane, Suite 500, Wilsonville, Oregon 97070; telephone (503) 682-5703.

Jillian Harvey-Asquith
Language Processors, Inc.
(508) 626-[illegible]

Bob Anundsen
88open Consortium, Ltd.
(503)692-3703

Kristie Graves
Cunningham Communication, Inc.
(408)982-0400

FOR IMMEDIATE RELEASE:

Language Processors, Inc. Announces Compiler Products for Motorola 88000 RISC Architecture

88open Consortium Task Force Drives Software Development

Framingham, Mass, February 21, 1989 -- Language Processors, Inc. (LPI) today announced software products for the Motorola 88000 RISC architecture and that it has joined the 88open Consortium Ltd. Software Initiative Program. The announcement was made at a news conference held by 88open and attended by more than 15 independent software vendors committed to the 88000 standard. LPI has been a member of the 88open Consortium since June 1988. Since July 21, 1988, Steven W. Weingart, Senior Vice President of Research and Development at LPI, has held a position on the 88open Board of Directors.

Language Processors, Inc. is porting its Family of Compilers to the 88000 RISC architecture. The LPI Family of Compilers includes: BASIC, C, COBOL, FORTRAN, Pascal, PL/1, and RPG II programming languages. Also included is CodeWatch, an interactive, source-level debugger. These products are expected to be released in the second quarter of 1989.
The Software Initiative was formed in August 1988 to promote the development of software for the 88000 RISC architecture. The program provides independent software vendors with access to a porting lab, 88000 development hardware, and marketing and technical support from the 88open Consortium. 88open Software Initiative sponsors include Data General, Dolphin Computer, a subsidiary of Norsk Data, Motorola, NCR, Opus Systems and Sanyo/Icon International.

"We have been working aggressively with independent software vendors to ensure that applications will be immediately available for 88000-based systems," said Bob Anundsen, Executive Director of the 88open Consortium. "The success and momentum of the program is directly related to the growing momentum for the Motorola 88000 architecture."

LPI's compilers and software tools are being developed to be compliant with 88open's Binary Compatibility Standard (BCS). The BCS has been adopted as part of the AT&T Application Binary Interface (ABI). It is expected that BCS compliant software will increase revenues for software developers and provide end users with shrink-wrapped software solutions for all 88000-based hardware systems.

"LPI will be the first language vendors to make available an entire family of integrated software development tools for the Motorola 88000 RISC environment," states Roy A. Finney, President of LPI. "These tools will give the end user the ability to establish proven software applications on the 88000 architecture, as well as develop new ones. This shows LPI's commitment to 88open and our high expectations of the business opportunities associated with Motorola's 88000 RISC microprocessor," he adds.

LPI, a producer of quality software development tools, offers a full suite of 32-bit programming languages, including BASIC, C, COBOL, FORTRAN, PASCAL, PL/I, RPG II, and an interactive source-level debugger, CodeWatch. These software products are presently offered on industry-standard systems such as Motorola's 680X0, WE32000, and Intel's 80386 running UNIX®, XENIX®, and MS-DOS® operating systems.

Headquartered in Framingham, Massachusetts, LPI is a market leader in quality language compilers and software development tools. Since its inception in 1980, LPI's technological advantage has attracted major customers including computer manufacturers, Fortune 500 companies, the federal government, application software developers, and a long list of domestic and international software distributors.

DataGeneral
DATA GENERAL CORPORATION
[illegible]

[illegible]

Upon Receipt
Andrew Mettinger
(508) 898-4060

DATA GENERAL UNVEILS FIRST INDUSTRY STANDARD FAMILY OF RISC-BASED SYSTEMS

SAN FRANCISCO, Feb. 28, 1989 -- Data General today unveiled the first members of its new family of products based on the Motorola 88000 architecture. The announcements include: a family of high-performance, VME-based systems that function as servers or multi-user systems for commercial and technical applications; a new family of desktop personal workstations; and DG/UX 4.1, the industry's only version of UNIX supporting a full RISC-based family of systems.

The 88000-based family establishes industry-leading price/performance standards for RISC-based systems. The systems perform at 20 or 40 MIPS or greater and the workstations perform at 17 or 20 MIPS or greater, based on Dhrystone measurements.

"Today's announcement establishes Data General as the industry's leader in cost-effective, distributed computing systems. Data General's experience in building large, well-balanced, high-throughput systems gives our emerging family of 88000-based products a major advantage over smaller machines from workstation and PC vendors. This experience has enabled Data General to design machines specifically for the newly emerging styles of computing such as the client/server model," said Herb Cohen, division director of product marketing.

"Today we are introducing the first members of a binary-compatible family of products that are faster, less costly and more flexible than any other systems on the market. For the price of today's personal computers, the 88000-based workstation delivers the performance of a mid-range workstation. The 88000-based system is not a headless workstation or a repackaged minicomputer. It is designed with sophisticated I/O and large capacity storage to support a large number of ASCII terminals, networked X-Window graphics terminals and workstations."

The powerful, midrange 88000-based system is available with one or two processors. By swapping the CPU board the user can upgrade from a single to a dual processor, nearly doubling CPU performance. The system is available in deskside and 14-inch-high, MEMA-Standard rackmount versions. Both systems include a ten-slot, industry-standard VME-compatible chassis, up to 208 MB of error-correcting code memory and cartridge and reel-to-reel tape options. The deskside version includes up to 2 gigabytes of mass storage and the rackmount version contains up to 16 gigabytes of mass storage through dual-ported 8-inch SMD disks.

Industry-standard wide and local area communications are provided by synchronous controllers (VSC/4) and the VME bus Ethernet controller (VLC). Up to 512 RS-232 asynchronous connections are handled by terminal services host adapter boards (VDA/128) which connect up to 128 asynchronous devices via cluster controller boxes.

The workstations can support from 4 to 28 MB of memory in a 2-1/2-inch-high pedestal enclosure. The workstations use a single multilayer board containing a Motorola 88100 CPU and FPU, two Motorola 88200 CMMUs (Cache and Memory Management Units), and monochrome or color graphics hardware co-processors. Interfaces for a PC/AT keyboard, a three-button optical mouse, a SCSI port, a parallel port, a serial port and 802.3 Ethernet are standard.

The monochrome workstations use the NEC uPD72120 Advanced Graphics Display Controller. The eight-plane color unit, based on a custom graphics coprocessor can display 256 simultaneous colors from a palette of 16.7 million colors. The 20-inch monochrome and 19-inch color flicker-free monitors operate at 70Hz and deliver 1280x1024 pixel resolution.

SCSI peripheral expension for the diskless workstations is via compact desktop housings. Up to three of these housings can be connected to a workstation. Peripheral options include a 160-MB or 322-MB 5-1/4-inch full-height Winchester disks, a 150-MB quarter-inch cartridge (QIC) tape drive and 5-1/4-inch or 3.5-inch floppy drive.

DG/UX 4.1 is an enhanced version of Data General's commercial-grade DG/UX 4.0 UNIX operating system and is designed to take full advantage of the Motorola 88000 RISC architecture. DG/UX 4.1 incorporates the major features found in UNIX systems with today's dominant UNIX standards.

"DG/UX 4.1, Data General's UNIX operating system with advanced symmetric multiprocessing, enables the 88000-based system to deliver cost-effective computing and modular growth while protecting investments in application software and peripherals," said Oeher. "DG/UX 4.1 conforms to the 88open Binary Compatibility Standard allowing customers to run all BCS-compliant software developed on 88000-based systems from any vendor. Application developers can use Data General's DG/UX 4.1 on all these systems to produce shrink-wrapped software for the entire 88000 marketplace."

DG/UX 4.1 is a full POSIX implementation, compatible with System V Interface Definition Issue 2 (SVID2) and Berkeley Software Distribution (BSD) 4.2. DG/UX offers the benefits of any standard UNIX system while providing unique, application-transparent capabilities such as fully symmetric multiprocessing support; a robust, flexible file system; an enhanced scheduler and extended error reporting. It also supports TCP/IP communications software, NFS 4.0 and X-Windows Version 11 Release 3.

DG/UX 4.1 fully implements the 88open Binary Compatibility Standard. "As a founding member of 88open Data General has actively participated in the 88open Software Initiative program designed to supply software for 88000-based systems," added Oeher. The 88open is a non-profit consortium of companies that manufacture, develop, sell or use products based on Motorola's 88000 RISC architecture.

MICRONICS RESEARCH CORP.

Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support

11 Feb 1989

Dear Don,

Just two typos to report in the February 'Logically Speaking' - both mine.

Page 19, 6th line up from bottom - should read "making"
Page 22, para 6, line 3, should read "internal-01 doesn't consistently line up ...", NOT "internal-23"

Now that I've got just over 2 years of programming in 68000 Assembler under my belt, I can see what a terrible price we have to pay by writing in position-independent code. I feel that programs such as my RBASIC would execute at least 2 to 3 times as fast if there were only a special version of SK*DOS dedicated to the single end-user (no multi-tasking or multi-user stuff in the works), something like the old 6809 FLEX. With RAMDISK at the top of memory, and a fixed Load-Address for applications programs, we could really blast along!

The Load-Address could be set high enough (say $10000) to allow for future improvements to SK*DOS. Ah, if only this could be so ... I'd gladly take the time to re-write RBASIC for such a set-up, and still keep up with the present version to run under a future multi-whatsit SK*DOS!! How about it, Pete? (Stark, that is)

Maybe readers who program in 68000 Assembler would be interested in learning of the many snags which lie in wait for the unwary, and some of the procedures I've evolved to make a programmer's life a little easier! Let me know!!

Don Williams,
68 Micro Journal,
5900 Cassandra Smith Road,
Hixson, TN 37343

Sincerely,

Bob

R. Jones
President

CROSS ASSEMBLERS

The CSC Cross Assemblers are actually a set of table-driven assemblers for several different microprocessors, currently including 1802/5, 6800/1/11/6303, 6804, 6805, 6809, 6502/3/65C02, 8080/5, 8048, 8051, Z-8, Z-80, with others being added occasionally. Separate 68010 and 32000 cross-assemblers are also available for the same price. A macro pre-processor is included with any purchase of cross-assemblers.

In addition to the basic ability to recognize the target computer's assembly language and to process symbolic address expressions, the cross-assemblers have the ability to support structured assembly programming, the ability to define conditional assembly directive expressions, and the ability to assemble programs stored in modular form.

Since the Motorola 6801 instruction set forms a superset of the 6800, 6802, and 6808 instruction sets, and is the same as the 6803 instruction set, the CSC 6801 Assembler is also capable of assembling programs written for all of those microprocessors.

The CSC 6801 Assembler is also capable of assembling programs for the Hitachi 6303, as it recognizes the six extra instructions, and for the Motorola 68HC11, as it recognizes the extra instructions and addressing modes.

The CSC 6809 assembler also recognizes 6800 and 6801 mnemonics, generating code for them to closely simulate the register and memory changes caused by the instructions on the 6800 and 6801.

The CSC 6805 Assembler is also capable of assembling programs written for the Motorola 146805 and 68HC05, as it recognizes the extra instructions implemented by those processors.

CSC also offers simulators to assist in the debugging of object programs
destined for 6800/1, 6805, 6809, 6502, and Z-80 systems, on selected host systems only.

The cross-assemblers, with printed manuals, binary utilities, and macro
pre-processor, are available for a retail price of $50 each $100 for three, or all for $200. The C sources are available for an additional $50 each, $100 for three, or $300 for all. The programs may be ordered from the following address:

Computer Systems Consultants, Inc.
1454 Laua Lane, Conyers, GA 30207
Telephone Number 404-483-4570/1717

Jim Howell
5472 Playa Del Ray
San Jose, Ca. 95123

Dear Don,

I have been using Introl C, version 1.5, for about three years on my OS-9 (tm Microware) Level 2 system (6809), and am quite pleased with it. It seems more bug-free than some of the MS-DOS C compilers. A few years ago, Dr. Dobb's Journal pointed out several bugs in various C compilers for MS-DOS. I tried all of these tests on the Introl C compiler and found that it has none of these bugs. I have, however, found one bug and a "problem" in the Introl compiler.

The bug occurs when a character (type "char") variable or expression is used as a subscript. In Listing 1, "tabl" is an array of structures, and the character variable "x" is used as a subscript. For the last line of the listing, the code generated by the compiler multiplies the subscript "x" by the size of the structure (10 bytes) to get the offset of the desired element of the array. In this example, the result will be 260 (decimal) or 0104 (hex). The high-order byte of this offset is then cleared (apparently because it is a "character" value!), resulting in 0004 (hex), or simply 4. The offset is then added to the address of the start of the array in order to point at the desired element of the array. The result is four bytes from the start of the array, and not 260 as it should be. Similar behavior will occur if an "int" (integer) array is subscripted with a character variable or expression whose value is 128 or more. This bug can be gotten around by using an "int" as the subscript or by using a cast, such as:

```
val = tabl[(int) x].a;
```

The "problem" mentioned in the first paragraph can be illustrated by considering the program "sample.c" (listing 2). To compile and link this program, one uses the commands:

```
icc sample
ilink sample
```

This results in an executable module called "sample", on the same disk as the source file. In my system, this is usually disk "/d1". I usually "load" a program first before running it, as follows:

```
load /d1/sample
sample
```

And now for the problem. When "sample" is run (the last line above), an access to the system disk is made before any output from "sample" occurs. After some investigation, I discovered what is happening.

When "sample", or any C program, is run, it starts in a "start-up" routine. This start-up routine does some initialization and set up, before calling "main". One of the functions of this initialization is to put in initial values for external data, which is all variables declared outside of all C functions. External variables which were declared with initial values will have those values inserted at this time. All other external variables will be set to 0 (zero).

In order to set up the declared initial values, the start-up code looks for a module whose name is the same as that of the executable module, with the characters ".DAT" added at the end. For this example, it would look for a module named "sample.DAT". This module, if it exists, is assumed to contain a table of initial values and where in the data area to put them.

First, the "link" system call is used, to see if the module is already in memory. If this call fails, the "load" system call is used, to look for the module in the commands directory. This is the disk access which I would like to avoid. If the "load" fails, too, then plugging initial values is skipped. Then "main" is called, which in this case prints a message.

This scheme works fine for programs that declare one or more initial values. In this case, the module sample.DAT (or whatever the name happens to be) is part of the executable file "sample". That is, if "sample.c" had contained one or more variables with an initial value, the file "sample" would contain an executable module called "sample" followed by the data module "sample.DAT". The "link" system call in the start-up code would succeed and no extraneous disk access occurs.

In the example, however, no initial values are given and the module "sample.DAT" is NOT part of the file "sample". In fact, "sample.DAT" is not created anywhere.

An easy "fix" would be to declare an initial value for some (external) variable. However, I preferred a solution that did not require an extra module, so I came up with the solution in Listing 3. The added function "fixload" patches the startup code to skip the attempt to load the module from the commands directory. It does this by looking at the function "cprep", the function in the introl library that looks for the data module. The first two tests in "fixload" check to see if the module has already been patched. The rest of the tests verify that the code being patched is really the intended code. If all of the tests succeed, a branch instruction is inserted. This branches around the attempt to "load" the data module.

With this fix, the program still looks for the module "sample.DAT" on disk the first time it is run, since the startup code is executed before "main" is called. After that, as long as the in-memory copy of the program is run, the extraneous disk access does not occur. To make the change more permanent, SAVE the

program (after running it at least once) and use VERIFY to correct the CRC. Then use ATTR on the corrected module to make it executable.

Note that the values used in "fixload" apply to the particular version of the Introl C Compiler that I have, and may or may not be correct for other versions.

The solution I have described is adequate for my use, but it requires the addition of code (the function "fixload") to most of the C programs that I write for OS-9. A more elegant solution would be to write a separate program that patches the disk copy of the module directly (and updates the CRC).

```
/*
   Listing 1
   Shows bug in Version 1.5
   of Introl C Compiler.
*/

struct {
    int a;
    int b;
    int c;
    int d;
    int e;
    } tabl[30];

main() {
    int val;
    char x;

    x = 26;
    val = tabl[x].a;
}
```

The compiler generates the following assembly output. The "clra" instruction should not be there.

```
_tabl   comm            300
                loc             0
_main:  leas            -3,s
                ldb             #26
          stb           2,s
                lda             #10
        mul
                clra
                leax    _tabl,y
                ldd             d,x
                std             0,s
                leas            3,s
                rts
                end
```

Listing 2

```
/*      sample.c    06/09/88    */

main() {
    printf("This is a sample program\n");
}
```


Listing 3

```
/*      sample.c    06/09/88            */
/*      with the function "loadskip"    */

main() {
    fixload();
    printf("This is a sample program\n");
}

/*
          loadskip.c  06/13/86  15:35
       This routine patches "cprep" so that it will not
      look on the disk for <module>.DAT when the program
       starts.
*/
int cprep();

fixload() {
    char *p;

    p = (char *) &cprep;
    if(p[58] == 0x64 && p[57] == 0x30
        && p[53] == 0x10 && p[54] == 0x26
        && p[55] == 0x00 && p[56] == 0x16
        && p[75] == 0x10 && p[76] == 0x27) {
            p[57] = 0x20;
            p[58] = 0x10;
            }
}
```

Classifieds — As Submitted - No Guarantees

Surplus Unused Motorola VME Modules & Electronic Solutions Enclosures for Sale at Discount

MVME133	CPU Module-68020, 1MB DRAM, 68881 FPP, 3 serial Ports, EPROM Sockets, VMEbus Interface	$675
MVME225-1	1MB DRAM Module, A32/D32 VMEbus Interface	$380
MVME320A	Winchester / Fllopy Controller	$490
MVME332	8 Channel intelligent Serial Communications Module	$675
Series 7	Electronic Solutions 7 Slot Desktop Enclosure, P1/P2 Backplane, 325W PS, Space for Winchester/Floppy/Tape	$695

Respond to: John Gannon, RPG, P.O. Box C12399, Ste 162, Scottsdale, Arizona 85267 Phone (602) 951-3373

***

S+ Memory Cards, CPU Cards, Hard Disks w/Controller Cards, I/O Cards, Cabinets, Power Supplies.
S/09 CPU Cards, Memory, I/O Cards, Controller Cards, Cabinets, Power Supplies
3-Dual 8" drive enclosure with power supply. New in box. $125 each.
5-Siemens 8" Disk Drive, $100 each.

Tom (615) 842-4600 M-F 9AM to 5PM EST

***

GMX-20 with 68881 Uniflex, utility pgms and C; 16 MHZ. This month only $1900. Unused.
Marc Tailsman MD, 30818 S Coast Hwy, S Laguna, CA 92677, MSG: 714 499 1877

ATARI & AMIGA CALL

As most of you know, we are very sensitive to your wishes, as concerns the contents of these pages. One of the things that many of you have repeatedly written or called about is coverage for the **Atari & Amiga™** series of 68000 computers.

Actually we haven't been too keen on those systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari & Amiga machines. That makes a difference. I feel that we are ready tostart someserious looking into a section for the Atari & Amiga computers. Especially so since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you) and rumored for the Amiga. Many of you are doing all kinds of interesting things on these systems. By sharing we all benefit.

This I must stress - *Input from you on the Atari & Amiga*. As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. Which is the way it has been for over 10 years. We need articles, technical, reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari & Amiga systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it, drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!

DMW

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEMC2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SETC5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630 FAX (615)842-7990

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

!!! Subscribe Now !!!
68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard ☐ VISA ☐

Card #_____ Exp. Date_____

For 1 Year ____ 2 Years ____ 3 Years ____

Enclosed: $_____

Name _____

Street _____

City _____ State ____ Zip _____

Country _____

My Computer Is: _____

Subscription Rates

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

***Foreign Surface: Add $12.00 per Year to USA Price.**

***Foreign Airmail: Add $48.00 per Year to USA Price.**

***Canada & Mexico: Add $9.50 per Year to USA Price.**

***U.S. Currency Cash or Check Drawn on a USA Bank !**

68 Micro Journal
5900 Cassandra Smith Rd.
POB 849
Hixson, TN 37343

Telephone 615 842-4600
Telex 510 600-6630

Reader Service Disks

Disk- 1	Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk- 2	Diskedit w/ inst & fixes, Prime, *Pnnod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3	Cbug09, Secl, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6	**Purchase Order, Index (Disk file indx).
Disk- 7	Linking Loader, Rload, Harkness.
Disk- 8	Crtest, Lanpher (May 82).
Disk- 9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign,Help,Date.Txt.
Disk-14	*Init, *Test, *Tenninal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong),CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven. Language Recognition Utility, Anderson March1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - 68MJ 2/87
Disk-33	Build the GT-4 Terminal - 68MJ 11/87 Condon.
Disk-34	FLEX 6809 Diagnostics, Disk Drive Test. ROM Test, RAM Test - 68MJ 4/88Korpi.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

8" disk $19.50
5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50
Overseas add: $4.50 Surface - $7.00 Airmail

68 MICRO JOURNAL

5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4600 - Telex 510 600-6630

K-BASIC™

The Only 6809 BASIC to Binary Compiler for OS-9 FLEX or SK*DOS
Even runs on the 68XXX SK*DOS Systems*

Hundreds Sold at Suggested Retail:

~~**$199.00**~~

- 6809 - OS-9™ users can now transfer their FLEX™ Extended BASIC (XBASIC) source files to OS-9, compile with the OS-9 version and run them as any other OS-9 binary "CMD" program. Much faster than BASIC programs.

- 6809 - FLEX users can compiler their BASIC source files to a regular FLEX ".CMD" file. Much faster execution.

- 68XXX - SK*DOS™ users running on 68XXX systems (such as the Mustang-08/A) can continue to execute their 6809 FLEX BASIC and compiled programs while getting things ported over to the 68XXX. SK*DOS allows 6809 programs to run in emulation mode. This is the only system we know of that will run both 6809 & 68XXX binary files.

K-BASIC is a true compiler. Compiling BASIC 6809 programs to binary command type programs. The savings in RAM needed and the increased speed of binary execution makes this a must for the serious user. And the price is now **RIGHT!**

Don't get caught up in the *"Learn a New Language"* syndrome - Write Your Program in BASIC, Debug it in BASIC and Then Compile it to a .CMD Binary File.

For a LIMITED time save over 65%... This sale will not be repeated after it's over! *

SALE SPECIAL:

$69.95

SPECIAL

Thank-You-Sale

Only From:

CPI **S.E. Media™**

5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

A Division of Computer Publishing Inc.
Over 1,200 Titles - 6800-6809-68000
FAX (615)842-7990

* K-BASIC will run under 68XXX SK*DOS in emulation mode for the 6809.

Price subject to change without notice.

PT-68000 SINGLE BOARD COMPUTER

The PT68K2 is Available in a Variety of Formats
From Basic Kits to Completely Assembled Systems

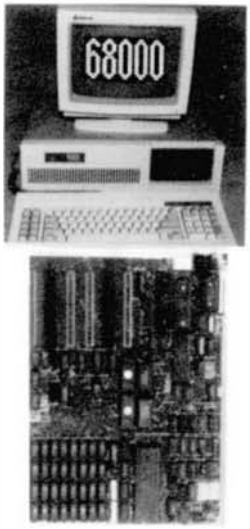
BASIC KIT (8 MHZ) - Board, 68000, HUMBUG MONITOR + BASIC in ROM, 4K STATIC RAM, 2 SERIAL PORTS, all Components **$200**

PACKAGE DEAL - Complete Kit with Board 68000 10 MHZ, SK*DOS, 512K RAM, and all Necessary Parts **$575**

ASSEMBLED BOARD (12 MHZ) Completely Tested, 1024K RAM, FLOPPY CONTROLLER, PIA, SK*DOS **$899**

ASSEMBLED SYSTEM - 10 MHZ BOARD, CABINET POWER SUPPLY, MONITOR + KEYBOARD, 80 TRACK FLOPPY DRIVE, CABLES **$1299**
For A 20 MEG DRIVE, CONTROLLER and CABLES **Add $295**

PROFESSIONAL OS9 **$500**



FEATURES

- MC68000 Processor, 8 MHZ Clock (optional 10,12.5 MHZ)
- 512K or 1024K of DRAM (no wait states)
- 4K of SPAM (6116)
- 32K,64K or 128K of EPROM
- Four RS-232 Serial Ports
- Floppy disk controller will control up to four 5 1/4", 40 or 80 track.
- Clock with on-board battery.
- 2 - 8 bit Parallel Ports
- Board can be mounted in an IBM type PC/XT cabinet and has a power connector to match the IBM type power supply.
- Expansion ports - 6 IBM PC/XT compatible I/O ports. The HUMBUG™ monitor supports monochrome and/or color adaptor cards and Western Digital winchester interface cards.

PERIPHERAL TECHNOLOGY

1710 Cumberland Point Dr., Suite 8
Marietta, Georgia 30067
404/984-0742
VISA/MASTERCARD/CHECK/C.O.D.

Send For Catalogue
For Complete Information On All Products

*SK*DOS is a Trademark of STARK SOFTWARE SYSTEMS CORP.
**OS9 is a Trademark of Microware

DATA-COMP SPECIAL

Heavy Duty Power Supplies

For A limited time our HEAVY DUTY SWITCHING POWER SUPPLY. These are BRAND NEW units. Note that these prices are less than 1/4 the normal price for these high quality units.

Make: Boschert
Size: 10.5 x 5 x 2.5 inches
Including heavy mounting bracket and heatsink.

Rating: in 110/220 volts ac (strap change) Out: 130 watts

Output: +5v - 10 amps
+12v - 4.0 amps
+12v - 2.0 amps
-12v - 0.5 amps

Mating Connector: Terminal strip
Load Reaction: Automatic short circuit recovery
SPECIAL: $59.95 each
2 or more $49.95 each
Add: $7.50 each S/H

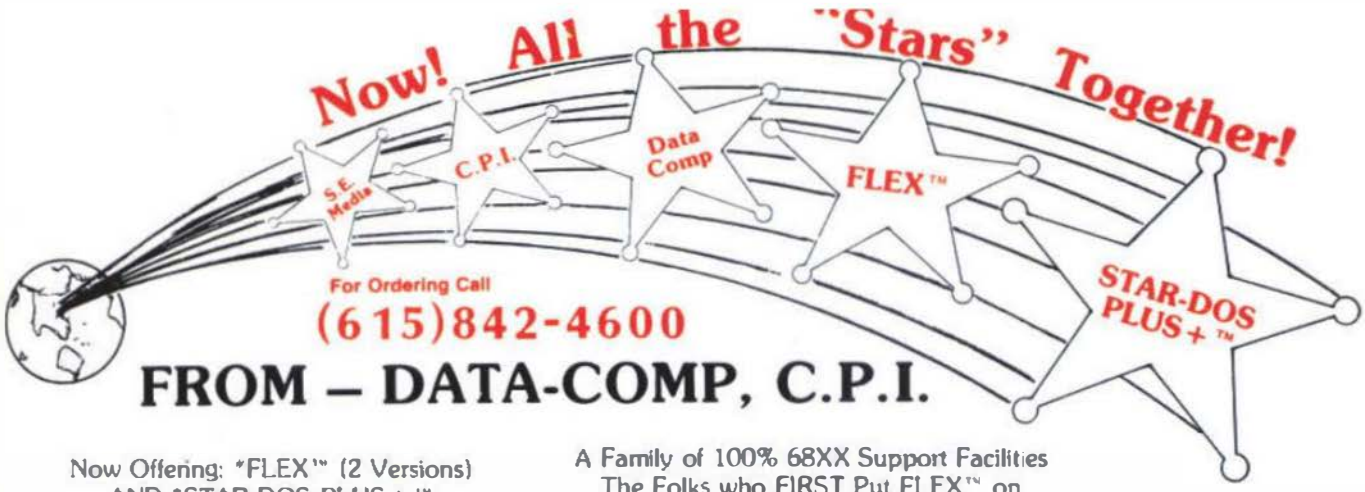
Make: Boschert
Size: 10.75 x 6.2 x 2.25 inches

Rating: 110/220 ac (strap change) Out: 81 watts

Outputs: +5v - 8.0 amps
+12v - 2.4 amps
+12v - 2.4 amps
+12v - 2.1 amps
-12v - 0.4 amps

Mating Connectors: Molex
Load Reaction: Automatic short circuit recovery
SPECIAL: $49.95 each
2 or more $39.95 each
Add: $7.50 S/H each

5900 Cassandra Smith Rd., Hixson, Tn. 37343 Telephone 615 842-4600 Telex 510 600-6630 Fax (615)842-7990



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

UPS Battery Backup

330	VA UPS 4 Outlets	309.00
520	VA UPS 4 Outlets	599.00
800	VA UPS 6 Outlets	859.00
1200	VA UPS 6 Outlets	999.00

Disks

5" Box of ten (10)

DS-DD	5.99	40 Track
DS-DD	14.95	80 Track

3.5 DS-DD $15.95

Printers

EPSON
LQ-500 24 Pin 350.00
LX-800 9 Pin 239.00

Switch Boxes

Serial/Parallel Converter	69.00
Parallel Serial Converter	69.00
Serial 2 Position Switch Box	27.00
Parallel 2 Position Switch Box	27.00
Serial 4 Position Switch Box	35.00
Parallel 4 Position Switch Box	33.00
Serial 4 Position Crossover Switch	44.00
Parallel 4 Position Crossover Switch	49.00
Serial 2 Position 9 Pin Switch Box	44.00

Gender Changer

SERIAL	
Male to Male	8.95
Female to Female	8.95
Centronics	
Male to Male	9.95
Female to Female	9.95

DATA-COMP
5900 CASSANDRA SMITH RD.
HIXSON, TN. 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN $3.50

VISA
master card

(615) 842-4600
FOR ORDERING
TELEX 5106006630

An Ace of a System in Spades! The New

MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the $500.00 OS-9 C compiler! This offer won't last forever!

NOT 128K, *NOT* 512K

FULL 768K No Wait RAM

The MUSTANG-08™ system took every hand from all other 68008 systems we tested, running OS-9 68K!

The MUSTANG-08 includes OS9-68K™ and/or Peter Stark's SK*DOS™. SK*DOS is a single user, single tasking system that takes up where *FLEX™ left off. SK*DOS is actualy a 68XXX FLEX type system *(Not a TSC product.)*

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed *whiz on disk I/O.* Fact is: the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks.

SAVED!

System includes OS-9 68K or SK*DOS - Your Choice

Specifications:

CPU	MC68008	12 Mhz
RAM	768K No Wait States	256K Chips
PORTS	4 - RS232 2 - 8 bit Parallel	MC68681 DUART MC6821 PIA
CLOCK	MK48T02	Real Time Clock Bat. B/U
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU Battery B/U - and $850.00 OS-9 Professional with C compiler included!

***$400.00** See Mustang-02 Ad - page 5 for trade-in details

MUSTANG-08

LOOK

		Seconds	32 bit Integer	Register Long
Other	68000 8 Mhz OS-9	68K	18.0	9.0
MUSTANG-08	10 Mhz OS-9	68K	9.8	6.3

```
Main()
{
                    /* int i; */
C Benchmark Loop    register long i;
                    for (i=0; i < 999999; ++i);
}
```

Now even faster! with 12 Mhz CPU

C Compile times: OS-9 68K Hard Disk	
MUSTANG-08 8 Mhz CPU	0 min - 32 sec
Other popular 68000 system	1 min - 05 sec
MUSTANG-020	0 min - 21 sec

25 Megabyte Hard Disk System

$2,398.90

Complete with PROFESSIONAL OS-9 includes the $500.00 C compiler, PC style cabinet, heavy duty power supply, 5" DDDS 80 track floppy, 25 MegByte Hard Disk - Ready to Run

Unlike other 68000 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable RDM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabytes of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC

MUSTANG-08 is a trademark of CPI

Data-Comp Division

CPI

A Decade of Quality Service"

Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

* Those with SWTPC Hi-density FLEX 5" - Call for special info.